(2)

RL-TR-93-133
Final Technical Report
July 1993

RESEARCH DIRECTIONS IN DATABASE SECURITY IV

MITRE

Rae K. Burns

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

Rome Laboratory
Air Force Materiel Command
Griffiss Air Force Base, New York

APPROVED: [illegible]

JOSEPH V. GIORDANO
Project Engineer

FOR THE COMMANDER: [illegible]

JOHN A. GRANIERO
Chief Scientist
Command, Control and Communications Directorate

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave Blank)	2. REPORT DATE July 1993	3. REPORT TYPE AND DATES COVERED Final Apr 91

4. TITLE AND SUBTITLE RESEARCH DIRECTIONS IN DATABASE SECURITY IV	5. FUNDING NUMBERS C - F19628-89-C-0001 PE - 35167G PR - 1068 TA - PR WU - OJ
6. AUTHOR(S) Rae K. Burns, Editor	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) MITRE Burlington Road Bedford MA 01730	8. PERFORMING ORGANIZATION REPORT NUMBER M92B0000118
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) Rome Laboratory (C3AB) 525 Brooks Road Griffiss AFB NY 13441-4505	10. SPONSORING/MONITORING AGENCY REPORT NUMBER RL-TR-93-133

11. SUPPLEMENTARY NOTES

Rome Laboratory Project Engineer: Joseph V. Girodano/C3AB/(315)330-3681

12a. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution unlimited.	12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

This report contains the papers written for the Fourth RADC Database Security Workshop, held in Little Compton, Rhode Island, 22-25 April 1991. The papers reflect a range of research topics, from the semantics of polyinstantiation to the probelms of actually building trusted database management systems. The goal of this workshop was to provide a forum for the discussion of the work in progress, to clarify and amplify ideas, and to advance the multilevel database security research agenda.

14. SUBJECT TERMS Multilevel Secure Database Management Systems	15. NUMBER OF PAGES 214
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT UNCLASSIFIED	18. SECURITY CLASSIFICATION OF THIS PAGE UNCLASSIFIED	19. SECURITY CLASSIFICATION OF ABSTRACT UNCLASSIFIED	20. LIMITATION OF ABSTRACT U/L

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. Z39-18
298-102

ACKNOWLEDGMENTS

The workshop was sponsored by the Air Force Rome Laboratory (formerly Rome Air Development Center (RADC)). Joe Giordano of Rome Laboratory provided the commitment and enthusiasm that made this fourth workshop a reality. The researchers who presented their work and joined in the discussions came from a variety of organizations, including Data Security, Inc., the Department of Defense, Gemini Computers, Inc., George Mason University, The MITRE Corporation, ONERA/CERT, Oracle Corporation, San Jose State University, Secure Computer Technology Corporation, and SRI International.

TABLE OF CONTENTS

SECTION 1

INTRODUCTION

The Fourth RADC Database Security Workshop was held at the Stone House Inn, Little Compton, Rhode Island, in April 1991. The workshop was the fourth in a series organized by Teresa F. Lunt of SRI International, with encouragement and support from Joe Giordano of Rome Laboratory. The workshop was sponsored by Rome Laboratory (formerly Rome Air Development Center (RADC)) and organized by Teresa F. Lunt of SRI and Rae Burns of The MITRE Corporation. The workshop participants came from the following organizations: Data Security, Inc., Department of Defense, Gemini Computers, Inc., George Mason University, The MITRE Corporation, ONERA/CERT, Oracle Corporation, San Jose State University, Secure Computer Technology Corporation, and SRI International.

During the workshop, researchers presented work in progress and discussed the theory and practice of multilevel database security. The "Workshop Summary" by Teresa F. Lunt highlights some of the technical issues addressed by the presentations and discussions at the workshop and provides a context for the remaining papers in this report.

Workshop Summary

Teresa F. Lunt
Computer Science Laboratory
SRI International
Menlo Park, California 94025

The fourth Rome Laboratory workshop on database security was held in Little Compton, Rhode Island, April 22-25. The workshop was organized by Teresa Lunt of SRI and Rae Burns of MITRE and was attended by 20 researchers active in the area of database security. This was the fourth such workshop sponsored by Rome Laboratory; the previous three were held in Menlo Park, California, in May 1988, in Bethlehem, New Hampshire, in May 1989, and in Castille, New York, in June 1990. The workshop participants presented work in progress and focused on some of the remaining challenging problems in database security. These included inference, polyinstantiation, conflicts between security and integrity, and secure concurrency controls.

The workshop opened with a session on the inference problem. The inference problem is when low data can be used to infer high data. Tom Garvey of SRI presented a paper coauthored with Teresa Lunt called "Characterizing and Reasoning about Inference Channels." He includes as part of the inference problem the inferences you might draw from aggregating information. In his talk Garvey focused on issues of detecting inference channels, quantifying the "width" of such channels, and eliminating channels. He proposed a formalization of the problem. He noted that some of the low data used to draw inferences is not stored in the database but is nevertheless known to users. Any practical approach to the problem has to take this into account.

Garvey described the inference methods deduction, induction, abduction, and probabilistic deduction, and he defined deductive, abductive, and probabilistic inference channels. *Deduction* allows you to reason directly from a set of information to a conclusion. *Induction* allows you to draw conclusions about a situation from a set of similar examples. *Abduction* is a reasoning method that allows you to find a plausible explanation for a piece of information Q; that is, to find what additional information or assumptions are needed, when combined with the available data, to explain Q. Abduction can also be used to create disinformation to mislead low users into drawing erroneous but innocuous inferences. ***Probabilistic deduction*** allows you to reason about how likely a statement is to be true. Such quantitative reasoning methods are important in estimating the size or severity of an inference channel.

Garvey described ***weighted abduction***, in which the cost of an inference is estimated. This can be used to characterize the risk of an inference channel. Costs are estimated by assigning costs to hypothesized assumptions and to inference steps. Thus, the greater the number of formulas assumed and the greater the number of inference steps, the greater the cost and the more difficult the inference. If the cost is sufficiently high, it could be argued that the high information is sufficiently safe from inference by low users. Similarly, Garvey described computing the probability that low users can infer high data. A probability channel exists of the probability exceeds some threshold value. Otherwise, the risk can be considered sufficiently small Garvey is investigating methods for combining probability with weighted abduction to compute risks for specific inferences.

Vicky Ashby of MITRE described her examination of many classification guides. She found that these guides don't provide the detailed guidance that would allow a data designer to determine the classification of the individual elements and tuples in a multilevel database. She also found some examples of inference problems in the guides. For example, individual cargo movements are not classified but the plan that determines them is classified. For some army systems she found examples of quantity-based aggregation, where, for example, greater than five records is secret whereas fewer are unclassified. In addition, any aggregate operation (e.g., sum or average) over greater than five records is also secret.

She presented some solutions that have been proposed for the latter example. All aggregate operators could be disabled for low users or for all users. The aggregate operators available to low users could be altered so that no more than five records can enter into a computation. *Ad hoc* queries could be prohibited, and predefined queries could be guaranteed to follow the classification rules. Bill Maimone of Oracle Corporation suggested auditing low users and raising an alarm if too many records are requested.

Some felt that the information being protected in this example is not the set of five or more records but the information that can be inferred form it. The approaches proposed by Garvey might yield better solutions to such problems.

LouAnna Notargiacomo of MITRE gave a presentation in which she raised issues of automatic classification using classification constraints. Classification constraints are rules that would be interpreted and enforced by an expert system or something similar. The use of classification constraints has been proposed to help users avoid making mistakes when entering data.

In a high assurance system, the use of classification constraints raises the question of how to verify the correctness of the rule interpreter and of the rulebase; the theory on which to base such a verification doesn't exist. Teresa Lunt noted that in addition you need to verify the rulebase against real-world policy, which is also a hard problem.

Notargiacomo asked what the safe use of classification constraints might be. For example, their use could be reserved for upgrade and downgrade decisions where a human is in the loop.

Notargiacomo noted that classification constraints constitute new mandatory policy that is added to the system *after* the system has been evaluated. Thus, the addition of rules that were not present in the original evaluation or changes to the rulebase may invalidate the previous evaluation or certification. In addition, updating the constraints may require reevaluating all the labels in the database.

Notargiacomo concluded that the risks associated with the use of classification constraints must be determined and issues of evaluation, certification, and accreditation must be addressed before classification constraints can be considered for use.

Rae Burns of MITRE thinks that classification constraints should be internal TCB data and not user-visible or labeled. She pointed out that the constraints are not used for labeling data if they are used as was proposed in SeaView. In SeaView, all data is written at the subject class, and the constraints merely prevent high users from overclassifying low data. Thus, if a high user attempts to enter data that should be low the operation is rejected. Burns also wants to reject the operation or upgrade the data if a low user enters data that should be high.

With Burns' proposal there are covert channels associated with the enforcement of the constraints. By observing what's accepted or rejected or disappears by upgrading, low users can infer the constraints, which may contain sensitive information (for example, all flights to Baghdad are top secret). A safer choice is to label the constraints, as is done in SeaView.

Tom Garvey noted that Burns is proposing to allow channels to exist as an engineering expedient. He feels this is dangerous unless you can quantify the risk of the inference channel.

In a session on polyinstantiation, Sushil Jajodia of George Mason University summarized the sources of polyinstantiation, as identified originally by SeaView. Polyinstantiation can arise through updates by high users or by low users. In the case of high users, it is possible to reject an update that would lead to polyinstantiation. However, in the case of low users, rejecting such an update is an inference channel. SeaView raised the possibility, which was later discarded, of allowing low users to overwrite high data rather than to polyinstantiate.

There is currently ongoing research at SRI and independently at George Mason University to define the behavior desired for the update operation when there is polyinstantiated data in a relation. Jajodia advocates matching the update predicate on all *visible* tuples, whereas SeaView has proposed to update *all* matching tuples. Bill Maimone pointed out that with Jajodia's proposal all the visible tuples aren't actually updated because some are polyinstantiated when doing otherwise would have violated the *-property. Maimone proposed an alternative in which the update is rejected if all matching tuples cannot be updated. If a high user wants to match on only high tuples, thereby ensuring that all matching tuples are updatable, the user could include "where label = S" in the update predicate. Lunt pointed out that an unsettling aspect of Jajodia's proposal is that it exhibits the same behavior whether or not the user includes "where label = S" in the predicate.

Jajodia presented yet another solution, in which a high user logs in at low and marks low data as 'restricted,' meaning that low users cannot update the data, thereby preventing low users from inadvertently polyinstantiating the data.

Chip Paradise of MITRE described the Air Force Military Airlift Command's approach to polyinstantiation in a system they are building. In that application, high data is deemed to have greater validity than low data. So in the case of polyinstantiation only the high data is returned to high users. Cover stories are required, for example, a polyinstantiated destination for the same mission ID. There is also a requirement for element-level labels and for uniformly classified attribute groups, as in SeaView. With uniformly classified attribute groups, all the elements in the group have the same classification within any given tuple.

In a talk called "The Trouble with Writeup," coauthored with Richard Allen of Oracle, Bill Maimone spoke about the integrity problems introduced by allowing subjects to write up. For example, if a low user inserts a high tuple with the same key as a preexisting high tuple, the tuple cannot be accepted, but the low user can't be informed of the key conflict. The tuple can't be polyinstantiated because one already exists at the same security level.

Referential integrity is enforced for insert, update, and delete statements and requires a writeup by a trusted subject, unless foreign keys are prohibited from referring to data at higher or lower security levels. Typical options for enforcing referential integrity include the "restrict" option (data can't be deleted if it is referenced) and the "cascade" option (if a reference is deleted, all referenced data is also deleted). For example, "on delete cascade" could be specified for a foreign key in the "create table" statement.

A *trigger* is code that executes on delete, update, or insert. For example, a trigger could be defined to execute when the delete command is used for the employees table to raise an exception if salary > 10000 and name = 'Allen' and to then delete a low department form the departments table. If there are both high and low employees, this trigger requires a trusted subject and results in illegal information flow. Thus, we can't let the average user create triggers. Even if triggers don't write low data based on the values of high data, they can result in covert channels if they are allowed to write up. This is because if a trigger fails, the transaction should abort. If the failure isn't reported to the user, you sacrifice integrity.

There are similar problems with integrity constraints. Constraints are invariants. If you can see a constraint then all the data you see should conform to it. If a constraint is added or modified, all the stored data must be corrected to conform to the new constraints. This can create covert channels if a constraint relates the values of high and low data or if constraints are classified higher than the table level. A constraint should be allowed to be classified higher than the table level only if it includes a predicate such as "where label > secret" and thus clearly applies only to high data. This points out the need to control who adds or modifies constraints.

Transactions pose similar problems. If a transaction writes to more than one security level, we can't allow the transaction level to float up, because we must return information at both levels about the failure or success of the transaction.

Trusted ORACLE gives users a variety of alternatives. For example, users can opt for either security or integrity on a table by table basis.

Maimone noted that users are demanding integrity features. He concludes that a strong concept of integrity implies the need to control writeup. He convincingly demonstrated that the conflicts between security and integrity are much more complicated than database designers are expecting.

Dick O'Brien of SCTC raised additional integrity issues concerning the deletion of multilevel relations. Should we allow a low user to delete a table if it contains high tuples? He suggested rejecting the operation or alternatively deleting only the high tuples and upgrading the table definition. He advocates restricting the "drop table" privilege to database administrators. Someone noted that Sybase took the "drop table" command out of its SQL and allows only system administrators to delete tables. In ORACLE, only the table owner can delete the table and must have read/write permission to *all* data in the table. This could result in a situation in which no one is authorized to delete the table, if the owner is not cleared for all the data in the table.

Cynthia Irvine of Gemini Computers described an approach Gemini is taking to evaluate a database system under the TNI rather than the TDI. With this approach, if the system component that enforces mandatory security is A1 and the other TCB components are C2+ (C2 assurance with some additional functionality), the overall system can receive an A1 rating under the TNI. Gemini is currently working on the GEMSOS M-component evaluation under the TNI.

Ravi Sandhu of George Mason University gave a talk called "Evaluation by Parts, or is the TDI Possible" in which he commented that our understanding of evaluation by parts is very primitive; the TDI won't be the last word on it. There are currently two notions of evaluation by parts: the TDI's layered composition, called TCB subsets, and the TNI's composition of disjoint components. The idea is that the various TCB components can be independently evaluated and composed into a system.

The TDI contains two varieties of TCB subsets, which it calls constrained and unconstrained. *Constrained TCB subsets* have no trusted subjects outside, whereas *unconstrained subsets* may. With unconstrained TCB subsets, evaluation by parts is problematic; the system as a whole must be reanalyzed with respect to penetration testing and covert channel analysis. This reflects the fact that the addition of a trusted subject to an evaluated system invalidates its rating.

With TCB subsets, each subset enforces its own security policy, and the "higher" subsets rely on the "lower" subsets for correct enforcement of their policies. For the TDI, typically the "higher" subset is the database system TCB and the "lower" subset is the operating system TCB. Prior evaluation of the lower subset tells you nothing about its correctness, for example, that user data stored is maintained correctly. Thus if the higher subset's policy enforcement data is stored as user data with respect to the lower subset, the system cannot guarantee the correct enforcement of the higher subset's policy. This applies to the unconstrained TCB subset approach to database systems, in which the multilevel database is stored in system-high files. Thus, in addition to flow properties, correctness properties of the lower subset must also be demonstrated to support evaluation by parts. Such a demonstration will not have been included in an Orange Book evaluation.

Ira Greenberg of SRI discussed how to provide secure concurrency control for multilevel databases without using trusted code. He showed that serializability is achievable under these constraints, but the solutions have undesirable properties (for example, high users get arbitrarily old versions of low data). He proposed new correctness criteria as follows: single-level serializability at each security level; single-level read consistency (consistency when reading related low data); progress (you should see a version at least as new as any version seen before); and recency (you see the most recent committed version). These provide security with most of the guarantees of serializability but with better performance.

In closing remarks, Joe Giordano of Rome Laboratory noted that the 1982 Summer Study agenda will be completed with the SeaView prototype delivery, expected soon. It is time to propose a new research agenda for the coming decade. This will be the topic for the fifth Rome Laboratory sponsored workshop, to be held in October 1992.

SECTION 2

APPROACHES TO INFERENCE CONTROL

The risk of unauthorized inference remains a major concern for multilevel database applications. In his paper "Aggregation and Inference Problems in Multilevel Secure Systems," Sushil Jajodia provides an overview of the inference problem and some of the approaches that have been proposed to reduce the risk of disclosure from inference. Garvey, Lunt, and Stickel explore a new perspective on inference through formalisms derived from artificial intelligence. Their paper "Characterizing and Reasoning about Inference Channels" describes the use of logical reasoning techniques to study several database inference problems. Finally, T. Y. Lin and Yazun Al-Eifan analyze an approach to aggregation in their paper "Entropy, Ordering, and Aggregation." This approach applies the concept finite entropy to provide randomness of ordering within aggregations.

AGGREGATION AND INFERENCE PROBLEMS IN MULTILEVEL SECURE SYSTEMS

Sushil Jajodia

Department of Information and Software Systems Engineering
George Mason University, Fairfax, VA 22030-4444

1. INTRODUCTION

A *multilevel* database management system (DBMS) contains data that are classified at different security levels. All database users are also assigned security clearances, and it is the responsibility of a multilevel *secure* DBMS to assure that all users gain access to only those data for which they have the appropriate clearance. The rules that govern how a secure DBMS controls access to data are known as the system's *security policy*.

All multilevel secure DBMSs that have been designed and are currently being built enforce a suitable interpretation of the DoD security policy, and in [1] DoD established a metric against which various computers systems can be evaluated for security. It developed a number of *levels*, A1, B3, B2, B1, C2, C1, and D, and for each level, it listed a set of requirements that a system must have to achieve that level of security. Briefly, the D level consists of all systems which are not secure enough to qualify for any of A, B, or C levels. Systems at levels C1 and C2 provide discretionary protection of data, systems at level B1 provide mandatory access controls, and systems at levels B2 or above provide increasing assurance, in particular against covert channels. The level A1 which is most rigid requires verified protection of data.

We will see below that the inference problems are different from covert channels. Thus, even if a DBMS meets all requirements for the level A1, it does not protect data from all information flows that violate the security policy.

In this paper, we will examine the various kinds of inference threats that arise in a multilevel secure database system. We will characterize the inference problem as well as survey methods that have been developed for dealing with it.

1.1. THE INFERENCE PROBLEM

In a DBMS, it is possible for users to to draw inferences from the information they obtain from the database. The inference could be derived purely from the data obtained from the database system, or it could additionally depend on some prior knowledge which was obtained by users from outside the database system. An inference becomes a problem in a multilevel secure system when more highly classified information can be inferred from less classified information. Therefore, a secure DBMS needs to provide *inference control:* Some item of more highly classified information cannot be inferred by combining less classified data.

There are two other related problems, aggregation problem and data association problem, which are defined as follows:

An *aggregation problem* occurs whenever there is a collection of data items that is classified at a higher level than the levels of individual data items by themselves. The classic example is that of an organization whose telephone directory in its entirety is classified while individual listings are unclassified.

A *data association problem* occurs whenever two values seen together are classified at a higher level than the classification of either value individually. As an example, the list consisting of the names of all employees and the list containing all employee salaries are unclassified, while a combined list giving employee names with their salaries is classified. Notice that the data association problem is different from the aggregation problem since what is really sensitive is not the aggregate of the two lists, but the exact association giving an employee name and his or her salary.

According to our definitions, the inference problem encompasses both the aggregation problem and the data association problem. Also, note that inference problems are different from covert channels since an inference requires an active agent at the low level, while a covert channel requires two active agents, one at the low level and the other at the high level, and an encoding scheme.

There are many difficulties associated with determining when more highly classified information can be inferred from less classified information. The biggest problem is that it is impossible to determine precisely what a user "knows." The problem is at least manageable if we adopt the *closed-world assumption* and assume that if information Y can be derived using information X, then both X and Y are contained in the database. By ruling out inferences that lie outside the database, the closed-world assumption provides a structured framework within which we can look for inference problems [10]. However, the outside knowledge that users have plays a significant role if we wish to take all inferences into account.

2. SOLUTIONS TO THE INFERENCE PROBLEMS

Inference control methods can be broadly classified into one of following three categories:

- *Inference Prevention Methods.* The methods in this category guarantee that inference problems cannot occur in the first place.
- *Inference Detection and Resolution Methods.* These methods detect potential inference problems in advance and take steps so they will not occur.
- *Limited Inference Methods.* Unlike the methods in the previous two categories, methods in this category allow inferences to occur, but these methods limit their bandwidth. These methods are useful in situations where

either the amount of inference is so small that it does not pose any threat or where the inference bandwidth is quite small, but it is too expensive or not desirable to close.

2.1. Inference Prevention Methods

2.1.1. Appropriate Labeling of Data and Integrity Constraints

If information x permits disclosure of information y, one way to prevent this is to reclassify all or part of information x such that it is no longer possible to derive y from the disclosed subset of x. To illustrate, suppose that an attribute A is *Unclassified* while attribute B is *Secret*. Suppose the database enforces the constraint $A + B \leq 20$ which is made available to *Unclassified* users. The value of B does not affect the value of A directly, but it does determine the set of possible values A can take. Thus we have an inference problem.

There are two ways to deal with the violations of this type. We have to reclassify either the data or the constraints appropriately. To this end, semantic database model techniques have been used to determine appropriate classification of data and integrity constraints [4, 12, 18]. Conventional data models (such as hierarchical, network, and relational data models) use overly simple data structures (such as trees, graphs, or tables) to model an application environment. Semantic database models, on the other hand, attempt to capture more of the meaning of the data by providing a richer set of modeling constructs. Since integrity and secrecy constraints can be expressed naturally in semantic database models (see [18]), they can be used to detect inference problems during the database design phase.

2.2. Inference Detection and Resolution Methods

Many inference violations arise as a result of a query which asks for the data which are at the user level, but its evaluation requires accessing data which are above the user's level. As an example, suppose that data are classified at the relation level, and that we have two relations, an *Unclassified* relation, called EP, with attributes EMPLOYEE-NAME and PROJECT-NAME, and a *Secret* relation, called PT, with attributes PROJECT-NAME and PROJECT-TYPE, where EMPLOYEE-NAME is the key of the first relation and PROJECT-NAME is the key of the second. (The existence of the relation scheme PT is *Unclassified.*) Suppose an *Unclassified* user makes the following SQL query:

```
SELECT  EP.EMPLOYEE-NAME
FROM    EP, PT
WHERE   EP.PROJECT-NAME = PT.PROJECT-NAME
```

If this query is evaluated by taking the natural join of the two relations, EP and PT, along PROJECT-NAME, and then projecting along EMPLOYEE-NAME, we have an inference problem, even though only the unclassified data (employee names) is being returned to the user. Although the output of this query is unclassified, it reveals classified information.

One method that can be used to inference violations of this sort is *query restriction*. If the system can ensure that all data used in the process of evaluating the query is dominated by the level of the user, then these inference violations cannot occur. To this end, the system can either modify the user query such that the query involves only the authorized data or simply abort the query.

Polyinstantiation is another technique that can be used to prevent inference violations [11]. To illustrate, suppose a low security-class user wants to enter a tuple in a relation in which the data are classified at the either the tuple or element level. Whenever a tuple with the same key at a higher security class already exits in the relation, both tuples are allowed to exist, but the security class is treated as part of the key. Thus the tuple entered at the low security class and the tuple entered at the higher security class will always have different keys, since the keys will have different security classes.

Finally, auditing can be used to control inferences. For instance, a history can be kept of all queries made by a user. Whenever the user makes a query, the history is analyzed to determine whether the response to this query when correlated with the responses to earlier queries could result in an inference violation. If a violation could arise, the systems can take appropriate action (for example, abort the query).

There is a side-benefit of this approach: it may deter many inference attacks by threatening discovery of violations. There are two disadvantages of this approach: One, it may be too cumbersome to be useful in practical situation. Two, it can detect a very limited types of inferences (since it is based on the hypothesis that a violation can always be detected by analyzing the audit records for abnormal behavior.)

2.3. Allowing Limited Inferences

So far we have mainly considered various means for eliminating all inferences. There are situations where it is possible to allow limited inferences. These methods are useful in those cases in which the inference bandwidth is so small that these violations do not pose any threat. Consider the following example. Suppose that data are classified at the attribute (column) level, and that we have two relations, one with the *Unclassified* attribute PLANE and *Secret* attribute DESTINATION, and another with the *Unclassified* attribute DESTINATION and *Unclassified* attribute FUEL-NEEDED. Suppose also that, although knowledge of the fuel needed for a particular plane can give information about the destination of the plane, there are too many destinations requiring the same amount of fuel for this to be a serious inference threat. Moreover, we do not want to go to the trouble of of clearing everybody responsible for fueling the plane to the *Secret* level. Thus we wish to make the derived relation with attributes PLANE and FUEL-NEEDED available to *Confidential* users.

Even though we have decided that this information does not provide a serious inference threat, we cannot allow users to perform the natural join and projection themselves, since it may provide a signaling channel. One solution is to use the "snapshot approach:" a trusted user creates a derived *Secret* relation with attributes PLANE and FUEL-NEEDED and then downgrade it to *Confidential*. Although this "snapshot" cannot be updated automatically without opening a signaling channel, it can be kept more or less up-to-date by having the trusted user re-create it from time to time.

A "snapshot" or a "sanitized file" is an important technique for controlling inferences, especially in offline, static databases. In particular, it has been used quite effectively by the United States Bureau of the Census [5,6,7].

3. CONCLUSION

A great deal of work has been performed in the inference area. Most of the research dealt with statistical databases (see [2] or [7]). More recently researchers have been looking into techniques to deal with the inference problem in databases. In addition to the references listed above, see also [3,8,10,9,13,14,15,16,17,19,20]. Although these methods can be extremely useful, a complete and applicable solution to the inference problem remains an elusive goal, and the inference problem should generally be viewed as being an open question.

References

1. "Department of Defense Trusted Computer System Evaluation Criteria," Department of Defense, National Computer Security Center, December 1985.
2. Nabil R. Adam and John C. Wortmann, "Security-control methods for statistical databases: A comparative study," *ACM Computing Surveys*, vol. 21, no. 4, pp. 515-556, December 1989.
3. Selim G. Akl and Dorothy E. Denning, "Checking classification constraints for consistency and completeness," *Proc. Symp. on Security and Privacy*, pp. 196-201, April 1987.
4. Leon J. Buczkowski, "Database inference controller," in *Database Security, III: Status and Prospects*, ed. D. L. Spooner and C. Landwehr, pp. 311-322, North-Holland, Amsterdam, 1990.
5. L. Cox, S. McDonald, and D. Nelson, "Confidentiality issues at the United States Bureau of the Census," *Jour. of Official Statistics,*, vol. 2, no. 2, pp. 135-160, 1986.
6. L. Cox, "Practices of the Bureau of the Census with the disclosure of anonymized microdata," *Forum der Bundesstatistik*, pp. 26-42, 1987.
7. Dorothy E. Denning, *Cryptography and Data Security*, Addison-Wesley, Reading, Mass., 1982.

8. Dorothy E. Denning, ''Commutative filters for reducing inference threats in multilevel database systems,'' *Proc. Symp. on Security and Privacy*, pp. 134-146, April 1985.

9. Dorothy E. Denning and Matthew Morgenstern, ''Military database technology study: AI techniques for security and reliability,'' SRI Technical Report, August 1986.

10. Dorothy E. Denning, ''A preliminary note on the inference problem in multilevel database management systems,'' *Proc. National Computer Security Center Invitational Workshop on Database Security*, June 1986.

11. Dorothy E. Denning, Selim G. Akl, Mark Heckman, Teresa F. Lunt, Matthew Morgenstern, Peter G. Neumann, and Roger R. Schell, ''Views for multilevel database security,'' *IEEE Trans. of Software Eng.*, vol. 13, no. 2, pp. 129-140, February 1987.

12. Thomas H. Hinke, ''Inference aggregation detection in database management systems,'' *Proc. IEEE Symp. on Research in Security and Privacy*, pp. 96-106, April 1988.

13. T. Y. Lin, ''Prababilistic measure on aggregation,'' *Proc. 6th Annual Computer Security Applications Conference*, pp. 286-294, December 1990.

14. T. Y. Lin, ''Multilevel database and aggregated security algebra,'' in *Database Security IV: Status and Prospects*, ed. Sushil Jajodia and Carl E. Landwehr, pp. 325-350, North-Holland, Amsterdam, 1991.

15. Catherine Meadows, ''Aggregation problems: A position paper,'' *Proc. 3rd RADC Workshop in Multilevel Security*, June 1990.

16. Neil C. Rowe, ''Inference-security analysis using resolution theorem proving,'' *Proc. IEEE 5th Int'l. Conf. on Data Engineering*, pp. 410-416, February 1989.

17. Gary W. Smith, ''Modeling security-relevant data semantics,'' *Proc. IEEE Symp. on Research in Security and Privacy*, pp. 384-391, May 1990.

18. Gary W. Smith, ''The modeling and representation of security semantics for database applications,'' Ph. D. Dissertation, School of Information Technology and Engineering, George Mason University, Spring 1990.

19. Bhavani M. Thuraisingham, ''Security checking in relational database management systems augmented with inference engines,'' *Computers & Security*, vol. 6, pp. 479-492, 1987.

20. Bhavani M. Thuraisingham. ''The inference problem in database security,'' Technical Report No. M89-52, Volume 3, The MITRE Corp., Bedford, September 1989.

Characterizing and Reasoning about Inference Channels

Thomas D. Garvey
Artificial Intelligence Center
SRI International

Teresa F. Lunt
Computer Science Laboratory
SRI International

Mark E. Stickel
Artificial Intelligence Center
SRI International

1 Introduction

In this paper, we attempt to relate certain formalisms used for reasoning problems in artificial intelligence (AI) to inferential security problems that arise in multilevel security for databases and knowledge bases. Inferential security remains one of the most critical and challenging problems to the database community and is destined to become more significant with the introduction of knowledge-based systems with their correspondingly richer relations among data and information. Here, we present a formalism for characterizing inferential problems of different types based on formal logical reasoning and theories for approximate reasoning. We believe the essence of inferential security problems are well captured by these formalisms.

2 Logical Formalisms for the Inference Problem

In multilevel databases, the inference problem is when some set of data classified at a low level (or *low data*) can be used to infer data classified at a high level (hereinafter referred to as *high data*). That is, there is a direct inference path (possibly including external data) from the low data to the high data. In order to develop automated methods for recognizing and eliminating inference channels, we require a formalization of the problem. The formalization suggested here is based on methods of formal logic and theories of approximate reasoning.

We characterize inferential security problems as belonging to one of three distinct types, based on the degree to which high data may be inferred from low data. The most restrictive type of channel occurs when a formal deductive proof of the high data can be derived from the low data—when this is the case, we say that a *logical inference channel* (or a *logical channel*) exists. A slightly weaker form of a channel is when a deductive proof may not be possible but a proof could be completed by assuming certain axioms. In this case, an abductive proof is possible, and we will term the channel an *abductive inference channel* (or an *abductive channel*). The final situation we will consider is when it is possible to determine likelihoods that assumed axioms might be knowable by a user with legitimate access to low data that would enable the inference of of high data with some measure of belief greater than an acceptable limit. In this case, we will (loosely) call the channel a *probabilistic inference channel* or just a *probabilistic channel.*

Logical channels can be described by standard propositional logic (PL) or first-order predicate logic (FOL). Most database systems may be modeled using PL while knowledge-based systems, which may allow for universally quantified expressions,[1] can be modeled using FOL (in this paper, we will use the terms *database* and *knowledge base* interchangeably). If PL is applicable, determining whether a logical channel exists is a decidable proposition but may be quite expensive. In the more general case of FOL, the question is not decidable: there is no way of knowing whether a logical channel exists until one is found. Since, in general, logical channels must not involve assumptions of facts, they must be based entirely on data found within the database.

Abductive channels represent a much more serious issue, since most inferential channels exist as the result of knowledge that a normal user might be expected to contribute to the problem but that is not an explicit part of the data or knowledge base. An abductive proof, however, can include assumptions and can consider the degree to which a user is likely to know some fact necessary to the completion of a proof. Since abduction involves assumptions about the user's belief structure, it involves modal logics, particularly epistemic logics.

A variety of schemes have been devised to determine the cost of an abductive proof [19, 20]. These typically include a cost (weight) for each additional proof step and a cost associated with an assumption. SRI has developed an abductive theorem prover (ATP) as an extension to Prolog that allows setting these weights as appropriate for the problem of interest. Setting assumption costs high relative to proof steps leads the ATP to prefer deeper proofs with fewer assumptions. Setting the assumption costs to infinity leads to standard theorem proving. Setting them low causes the ATP to prefer assumptions.

One means of setting these costs is to consider the likelihood that a particular user might know the assumed facts. Assumption costs could be related to these likelihoods, and the overall cost of the proof would then be a function of these probabilities.[2] A variety of computational schemes, based on classical probabilities [12], belief functions [13, 8], or fuzzy logic [14], could be considered for the task of determining the cost of an abductive proof incorporating beliefs. Using a formal theory for approximate reasoning would allow the computed cost to reflect the likelihood that high data could be inferred by a low user with ordinary or particular knowledge.

[1]Such as rules of the form, FOR ALL x, $p(x)$ IS TRUE. For example, ALL BIRDS CAN FLY.

[2]Such probabilities over states of belief are often referred to as *epistemic probabilities* [8].

In the remainder of this paper, we first present a general approach to the inference problem in multilevel databases, then we provide some background information on abductive and approximate reasoning, and finally we show how these techniques may be applied to detecting potential inference problems.

3 Inference Using External Data

Many inference problems can be solved through appropriate design and classification of the data structures in the database [4]. For example, consider a database containing personnel information, including names, addresses, and salaries of employees. Suppose that names, addresses, and salaries are all UNCLASSIFIED, but that names and salaries taken together are SECRET — that is, the association of a salary with an individual is SECRET. This problem is easily solved by the following data design

EMPLOYEES(EMP#, NAME, ADDRESS)
SALARIES(S#, SALARY)
EMP-SALARIES(EMP#, S#)

where the EMPLOYEES and SALARIES relations are both UNCLASSIFIED but the EMP-SALARIES relation is SECRET. Because there is no way to meaningfully join EMPLOYEES and SALARIES to match a salary with a name, the sensitive relation, represented by the relation EMP-SALARIES, is protected from uncleared users.

Now suppose that we have the above "safe" set of relations, and we want to add a new attribute, employee start date (which is not sensitive), to the database. It seems to make sense to add it to the SALARIES relation, as follows:

EMPLOYEES(EMP#, NAME, ADDRESS)
SALARIES(S#, SALARY, START-DATE)
EMP-SALARIES(EMP#, S#)

However, an employee's start date is an easily observable or discoverable attribute of an employee. Thus, an employee's identity may be inferred (or partially inferred) from his or her start date. This means that the sensitive relationship between an employee's name and his or her salary may be compromised. This problem can be removed in several ways. One way is to add the attribute START-DATE to the EMPLOYEES relation rather than to the SALARIES relation. Another way is to create a separate (unclassified) relation HIRE-DATES(EMP#, START-DATE). Note that EMP# rather than S# must be used as the key for the new relation, otherwise the inference is not removed. So we need to know what attributes act as such "almost keys" (also termed *identificates* [11]), or attributes that allow partial inference of the individual identifying information, and understand how these contribute to making inferences.

The fact that the SECRET relationship between employee name and salary is inferrable can be determined through analysis of the data structures and security constraints that would normally be part of a multilevel database. However,, inference problems cannot always be detected using only the information stored in the database. For example, to solve the second problem above, we need to know that employee name can be inferred from START-DATE, a piece of information that would not normally be stored in the database.

4 Abductive Reasoning

Here we present a brief overview of abduction, a form of reasoning we believe to be particularly appropriate for formulating the database inference problem and for designing automated tools for detecting inference channels. Abduction has traditionally been applied to diagnostic tasks [15, 16, 17, 18] that reason from events to causes. If Q is observed and needs to be explained and $P \supset Q$ is known, then P can be offered as a possible explanation of Q.

Deductive reasoning is the process of demonstrating that a formula is a consequence of a theory: $T \vdash Q$. *Abductive reasoning* is a distinctly different form of reasoning from deduction and is not limited to demonstrating that a formula is a consequence of a theory. In abductive reasoning, the objective is to find assumptions A such that $T \cup A \vdash Q$ even though Q may not be provable from T alone.

A large difference between abductive and deductive reasoning is the desire to discover the *best* abductive explanation. Many sets of assumptions may explain a phenonomenon, but some are preferable. Preferred explanations must often be chosen in an application-dependent, heuristic way.

In using an ATP for inference channel detection, high facts would become theorems to be proved. The ATP would back-chain through inference rules to low data (which would become the proof axioms) or to assumptions. No assumption would be permitted that was already present as a high fact. Acceptable assumptions proposed for a proof would need to be evaluated by the database security manager to determine the degree to which they may be known to low users.

To control the depth of the proof, we can provide a cost structure that allocates a cost value to each proof step and a cost to assumptions. By making the assumption cost high relative to the proof-step cost, we make the theorem prover favor deeper proofs. Alternatively, low assumption costs lead to relatively shallow proofs. From an informal point of view, an ATP used for detecting inference channels should have a cost for proof steps chosen to cause it to search moderately deeply for logical channels (i.e., channels that do not require assumptions), but not too deeply. The deeper the proof required, the more work a user will have to put into the deduction, and therefore, the less likely (or the lower the bandwidth of) the channel.

5 Abductive Reasoning about Inference Channels

In the above example, a naive data designer may have defined the data structures as follows.

EMPLOYEES(EMP#, NAME, ADDRESS)
SALARIES(S#, SALARY, START-DATE)
EMP-SALARIES(EMP#, S#)

Thus, each employee a has the following attributes:

$name(a)$
$address(a)$
$salary(a)$
$start\text{-}date(a)$
$emp\#(a)$ (database record identifier)
$s\#(a)$ (database record identifier)

The database is organized in an effort to keep $salary(a)$ secret. This is done by associating employee records identified by $emp\#(a)$ and salary records identified by $s\#(a)$ in the secret relation EMP-SALARIES.

The following functions, among others, are determined by the unclassified EMPLOYEES and SALARIES relations.

$$\begin{aligned}
employees_{12}(u) &= \{v \mid \exists w \in \text{EMPLOYEES}(u, v, w)\} \\
employees_{13}(u) &= \{w \mid \exists v \in \text{EMPLOYEES}(u, v, w)\} \\
employees_{21}(v) &= \{u \mid \exists w \in \text{EMPLOYEES}(u, v, w)\} \\
salaries_{12}(x) &= \{y \mid \exists z \in \text{SALARIES}(x, y, z)\} \\
salaries_{13}(x) &= \{z \mid \exists y \in \text{SALARIES}(x, y, z)\} \\
salaries_{31}(z) &= \{x \mid \exists y \in \text{SALARIES}(x, y, z)\} \\
salaries_{32}(z) &= \{y \mid \exists x \in \text{SALARIES}(x, y, z)\}
\end{aligned}$$

The functions $employees_{12}$, $employees_{13}$, $salaries_{12}$, and $salaries_{13}$ are singleton-set valued, since their domains are unique primary keys for the EMPLOYEES and SALARIES relations. The expression $employees_{21}(name(a))$ identifies a set of possible values of $emp\#(a)$ and $salaries_{31}(start\text{-}date(a))$ identifies a set of possible values of $s\#(a)$.

The security restriction on the database is violated if an unclassified user can infer elements of the EMP-SALARIES relation. We would like to be sure the statement $\neg$**Know** $emp\text{-}salaries(u, x)$ is true. A proof of **Possibly Know** $emp\text{-}salaries(u, x)$ would prove that security might be violated.[3] An abductive proof would create sets of assumptions that are sufficient to prove that security might be compromised if the assumptions are true. These assumptions can then be evaluated for plausibility. An abductive proof can be generated by backward chaining from the formula we wish to prove and treating as assumptions formulas we cannot prove.

[3]Modal logic operators like **Possibly** and **Know** are necessary to adequately express uncertainty and knowledge. For example, it is necessary to distinguish between the similar expressions $\exists y$ **Know** $salary(a) = y$ (I know a's salary) and **Know** $\exists y\ salary(a) = y$ (I know a has a salary). These sorts of distinctions are difficult to make in classical logic.

We give here an informal proof that elements of the intended-to-be-secret EMP-SALARIES relation might be knowable.

If the sets of possible values for $emp\#(a)$ and $s\#(a)$ are small, then **Possibly Know** $emp\text{-}salaries(u, x)$, where $u = emp\#(a)$ and $x = s\#(a)$. Formally,

$$\begin{aligned}
&\exists U\ small(U) \land \textbf{Know}\ emp\#(a) \in U \land \\
&\exists X\ small(X) \land \textbf{Know}\ s\#(a) \in X \\
&\supset \textbf{Possibly Know}\ emp\text{-}salaries(emp\#(a), s\#(a))
\end{aligned}$$

Thus, for the goal

$$\textbf{Possibly Know}\ emp\text{-}salaries(emp\#(a), s\#(a))$$

we have the subgoals:

$$\begin{aligned}
&\textbf{Know}\ emp\#(a) \in U \\
&\textbf{Know}\ s\#(a) \in X \\
&small(U) \\
&small(X)
\end{aligned}$$

The $employee_{21}$ and $salaries_{31}$ functions can produce sets U and X that satisfy the first two subgoals, yielding the following set of subgoals:

$$\begin{aligned}
&\textbf{Know}\ name(a) = n \\
&\textbf{Know}\ start\text{-}date(a) = s \\
&small(employee_{21}(n)) \\
&small(salaries_{31}(s))
\end{aligned}$$

At this point our abductive proof is complete. We have proved that if an employee's name and start-date are known, and if the number of possible EMP# and S# values computed from them is small (and certainly if they are both singleton sets), then we might know EMP-SALARIES(EMP#,S#) in the secret relation. Because these are all plausible assumptions, we must conclude that the database design is not secure.

The problem is even easier if formalized differently. Suppose instead we merely wish to keep secret the attribute $salary(a)$. It can be abductively proven that a's salary might be known ($\exists y$ **PossiblyKnow** $salary(a) = y$) if a's start-date is known and $salaries_{32}(start\text{-}date(a))$ is a small set.

This simple example illustrates the general concept presented here. Our investigations of this formalism will, we hope, lead to the development of database design tools so that a proposed database design can be analyzed for inference channels and restructured so that the problems are eliminated or minimized.

6 Approximate Reasoning

Inference problems arise when it is possible for a user to use low data to infer the truth of high data with some degree of probability. For example, flight-destination airports may be sensitive data, while aircraft range, payloads, and departure fields may be stored at a low security level. By combining information about range, payloads, and departure fields, a user may be able to greatly narrow the set of possible destination airports, and in so doing increase the *likelihood* that an aircraft's destination is among the reduced set. Further information (say, data about the aircraft-handling capabilities of the airfields in the reduced set) may serve to reduce the space of possibilities even more.

Such probabilistic channels are related to abductive channels because the assumptions and logical rules used in an abductive proof may have degrees of belief associated with them which represent the likelihood that they may be known to a user. These degrees of belief can then be propagated through the abductive proof tree to determine the degree to which the user is likely to be able to infer the high data in question. In effect, the ATP can be used to uncover the existence of a channel and approximate reasoning methods used to evaluate the relative seriousness of the channel.

Several schemes have been proposed to address the problem of *quantifying* the risk to security. Early work [5, 6] took the approach of characterizing the inferential closure of a core of unclassified information, with the aim of determining whether any classified information fell within the closure. When this occurs, an inference channel exists. The inferential closure includes all statements for which the relative change from the prior likelihood for the statement (modeled by its entropy) given statements in the unclassified core is greater than some present threshhold. The threshhold is a parameter that will determine the size of the closure. This work, while theoretically appealing, has proven impractical to implement.

An approach based on Bayesian probability [1] has been proposed as a more practical approach to estimating the security risk due to partial inferences. This approach is appealing except that it requires a great deal of probabilistic information, which is typically quite hard to estimate precisely [2, 8, 7, 9, 10]. Evidential reasoning (ER), based on the Shafer-Dempster Theory of Evidence, a non-Bayesian uncertainty theory, offers an alternative formalization that avoids these difficulties [13, 7].

6.1 Evidential Reasoning for Partial Inference

Evidential reasoning departs from classical probability theory in that it permits beliefs to be attached to disjunctions of statements, rather than requiring they be assigned to singletons in the universe of discourse (the set of mutually exclusive and exhaustive statements that form the "vocabulary" for the problem statement). In ER, this mutually exclusive and exhaustive set of concepts of interest is called the frame-of-discernment (or just the frame) and is designated Θ. Propositions are made up of elements of the set of all subsets of Θ, indicated by 2^{Θ}. Belief can be assigned to any proposition, including Θ itself; any belief assigned to Θ expresses total ignorance to that extent. An *evidential mass function* (or just mass function) represents the distribution of a unit of belief across selected (focal) elements of 2^{Θ}. A *body of evidence* is the frame of discernment and a particular mass function.

For example, we may know that a particular aircraft, because of its range and location, may be able to fly to a set of airports. When considering which airport it is really going to fly to, we can identify the destination only as a member of this set. Therefore, we may assign our belief about the plane's destination to the *set* of possibilities. In a Bayesian framework, we would be forced to distribute this belief to the individual elements of the set (thereby introducing new "information" into the process); in evidential reasoning we are not forced to make this distribution. This approach avoids the need for assumptions for values of missing data. When beliefs of components are later needed, they are underconstrained as a result of the disjunction, and an interval representation is needed to capture the true constraints. This interval enables the explicit modeling of both what is known (although with uncertainty) and what is unknown. The approach is valuable for representing human expertise where the available knowledge is likely to be imprecise and not well modeled by precise probability values.

Evidential reasoning provides an advantage over standard probability mechanisms in that prior probability distributions are not required in order to compute the likelihood resulting from combined bodies of evidence. This information is often unavailable in any event, but when it is available, evidential reasoning treats it as any other knowledge source.

For inference control, an abductive proof structure combined with information about the likelihood that a user might know facts assumed in the proof can be used to calculate the likelihood that the user could infer high data. Furthermore, sensitivity analyses can be carried out over the information structure in order to determine which information has had the greatest impact on the inference. This information might then be an initial candidate for upgrading in order to eliminate the channel.

Evidential reasoning techniques have been automated in a system called *Gister* [3].

7 Summary

We have formulated the inferential security problem as a problem in logical reasoning, specifically as a problem of attempting to determine whether there exists an inferential chain from low data to high data, possibly involving low data and inference rules that are not explicitly represented in the database itself. We have further suggested a new taxonomy of inference channels — the logical channel, the abductive channel, and the probabilistic channel — based on the reasoning process most appropriate to the particular inference problem. The application of abductive reasoning also offers a computational mechanism for detecting inference channels in databases. We feel that, as a logical formalism, abduction is the most appropriate model for most inference channels involving strictly logical inferences. We identified probabilistic channels as another important class of inference channels, those associated with the likelihood of inferring high data from low data that a user might be likely to know with some probability. We offer evidential reasoning as a candidate technology that could be linked with abduction to provide an effective computational framework for reasoning about such probabilities.

The formalisms proposed here can lead to a better understanding of the inference problem itself and to potentially practical solutions.

References

[1] L. J. Buczkowski. Database inference controller. In D. L. Spooner and C. Landwehr, editors, *Database Security III: Status and Prospects*, North-Holland, 1990.

[2] T. Y. Lin. Probabilistic measure on aggregations. In T. F. Lunt, editor, *Research Directions in Database Security, II*. SRI International, Menlo Park, California, 1989.

[3] J. D. Lowrance, T. D. Garvey, and T. M. Strat. A framework for evidential-reasoning systems. In *Proceeding of the National Conference on Artificial Intelligence*, pages 896–903, American Association for Artificial Intelligence, 445 Burgess Drive, Menlo Park, California 94025, August 1986.

[4] T. F. Lunt. Aggregation and inference: Facts and fallacies. In *Proceedings of the 1989 IEEE Symposium on Research in Security and Privacy*, May 1989.

[5] M. Morgenstern. Security and inference in multilevel database and knowledge-base systems. In *Proceedings of the ACM International Conference on Management of Data (SIGMOD-87)*, May 1987.

[6] M. Morgenstern. Controlling logical inference in multilevel database systems. In *Proceedings of the 1988 IEEE Symposium on Security and Privacy*, April 1988.

[7] E.H. Ruspini. Imprecision and uncertainty in the entity-relationship model. In C.V. Negoita and H.E. Prade, editors, *Fuzzy Logic and Knowledge Engineering*, pages 3–17. Verlag TÜV Rheinland, Cologne, 1986.

[8] E.H. Ruspini. Epistemic logic, probability, and the calculus of evidence. In *Proc. Tenth Intern. Joint Conf. on Artificial Intelligence*, Milan, Italy, 1987.

[9] E.H. Ruspini. The logical foundations of evidential reasoning. Technical Note 408, Artificial Intelligence Center, SRI International, Menlo Park, California, 1987.

[10] E.H. Ruspini. The combination of dependent evidence. Unpublished, 1989.

[11] G. W. Smith. Modeling security-relevant data semantics. In *Proceedings of the 1990 IEEE Symposium on Research in Security and Privacy*, May 1990.

[12] Judea Pearl. "Fusion, Propagation, and Structuring in Bayesian Networks," Tech. Report CSD-850022, Cognitive Systems Laboratory, Computer Science Department, University of California, Los Angeles, June 1985.

[13] Glenn A. Shafer. *A Mathematical Theory of Evidence*, Princeton University Press, Princeton, New Jersey, 1976.

[14] Lotfi A. Zadeh. "Fuzzy Sets as a Basis for a Theory of Possibility," *Fuzzy Sets and Systems*, Vol. 1, 1978, pages 3–28.

[15] P.T. Cox and T. Pietrzykowski. Causes for events: their computation and applications. *Proceedings of the 8th Conference on Automated Deduction*, Oxford, England, July 1986, pages 608–621.

[16] P.T. Cox and T. Pietrzykowski. General diagnosis by abductive inference. *Proceedings of the 1987 Symposium on Logic Programming,* San Francisco, California, August 1987, pages 183–189.

[17] H.E. Pople, Jr. On the mechanization of abductive logic. *Proceedings of the Third International Joint Conference on Artificial Intelligence,* Stanford, California, August 1973, pages 147–152.

[18] R. Reiter. A theory of diagnosis from first principles. *Artificial Intelligence 32,* 1 (April 1987), pages 57–95.

[19] M.E. Stickel. A Prolog-like inference system for computing minimum-cost abductive explanations in natural-language interpretation. *Proceedings of the International Computer Science Conference '88,* Hong Kong, December 1988, pages 343–350.

[20] M.E. Stickel. Rationale and methods for abductive reasoning in natural-language interpretation. *Proceedings of the IBM Symposium on Natural Language and Logic,* Hamburg, West Germany, May 1989.

ENTROPY, ORDERING AND AGGREGATION

Yazun Al-Eifan* and T. Y. Lin**

*Computer Science,Illinois Institute of Technology, Chicago, Ill 60616
**Math & Computer Science, San Jose State University, San Jose, CA 95192

1. INTRODUCTION

Defending against human reasoning has drawn considerable amount of interests in the community of database security. [Denn86a-b], [Denn87a-b], [Denn88a-b], [Dill88], [Hink88], [Lin89c], [Lin90a,d], [Lin91a,c], [Lunt89a], [Garv91], [Stac88], [SuOz87], [SuOz89]. [Thur87], [Thur90]. These works have been concentrated on the reasoning that comes from formal logic (e.g., deduction and abduction). In this paper, we enter into the territory of defending against the so called plausible reasoning [Poly68] or "educated guess" in laymen's term.

Aggregation exists if the aggregation of data has a higher sensitivity than each of the components considered separately. The famous phone book problem is the classical example; the phone number of an individual is not sensitive, but the entire phone book is sensitive. In this paper, we consider another type of aggregation problem in which the collection of the individual phone number in "random order" may not be sensitive, but the entire phone book in "particular order" is highly sensitive. According to the current definition, this is an aggregation problem; however, this type of aggregation may not be the examples that was intended to be covered.

In [Lunt89], Lunt gave a concise illustration on various problems surrounding the notion of storing the sensitive association high. "The sensitive association will be compromised if the employee and salary records are both sorted in the same order ... need to be further studied." Here, she has clearly spelled out the importance of the ordering. If the database security community decides to store association high and data low, then this paper offers a possible theory to accomplished it. Present theory has its own interests, it is closely related to the problem of encryption. We will report on this in near future.

----

* This paper contains part of Ph. D. dissertation of first author at IIT supervised under J. Kenevan and T.Y.Lin.

Classically, there are two type of measures on the ordering. One is the entropy, another is Kolmogorov complexity (Algorithmic information theory). Due to Chaiten's Godel type incompleteness results [Chai87a,b], most likely there will not be able to use Kolmogorov's complexity directly as a measurement for real life problems. So entropy is our choice. There are various form of entropy. In statistical mechanics, information theory, entropy is defined in terms of probability. While in ergodic theory, measure theory are used to define the entropy (Probability is a special case of measure theory). In (topological) dynamical systems, one defines entropy in terms of open covering. In our applications, there is no natural probability in our setting, but there is a natural measure, namely the counting measure. So we adopt the entropy in ergodic theory as our guiding example. The entropy in ergodic theory is designed to study the infinite mathematical world, we have to adopt it to the finite world. In [Lin90], we have some preliminary report on such entropy -- call it finite entropy. In terms of finite entropy, we propose "quasi Kolmogorov randomness" as an alternative for Kolmogorov randomness. However, the main theme of this paper is on the computation of the entropy. From our computation, finite entropy seems capture the intuitive notion of randomness.

The permutation below is the result of permuting 1,2,3,4,5.

```
Permutation:1,2,3,4,5; Entropy=  2.0250 - No moves
Permutation:1,2,3,5,4; Entropy=  1.1676 - 2 moves
Permutation:1,2,5,3,4; Entropy=  0.4609 - 3 moves
Permutation:1,5,2,3,4; Entropy=  0.1723 - 4 moves
Permutation:4,5,1,2,3; Entropy=  0.0000 - 5 moves
```

This paper is a coninuation of previous report on finite entropy [Lin90], for completeness, we will present this paper as a complete paper, readers who accepted the notion of ordered aggregation or familiar with [Lin90] may skip part of Section 2 (motivational examples) and 3 (important issues in the ordered aggregations), and go directly to the computation.

2. EXAMPLES

Example 2.1

Message LOC:

"The location of the essential information of the secret particle beam weapon is indicated in the painting of the statue of liberty hanged on the west wall of the east room of the south wing of the north complex of the state building".

The message LOC is sensitive, however, its alphabetical order is hardly sensitive. In general, a collection of words (or characters) in "wrong" order does not carry the same sensitive information.

This message is an aggregation problem, since each word (or character) is obviously unclassified, but the whole collection (in some order) is highly sensitive. However, it is quite different from the usual aggregation, because the "wrong" order is not necessary sensitive. We call such non-commutative type of aggregates ordered aggregates.

To convince readers and ourselves, let us examine the effect of classical solution to this problem.

(a) SeaView's solution [Lunt69]: Each individual words is upgraded to the text level.

(b) LDV's solution: Part of the text can be released, however, the whole text will be protected.

(c) Hinke's solution: Some words will be upgraded to the level of the text.

(d) Lin's solution [Lin89,90a,b]: No users are allowed to see any individual word, same effect as SeaView.

(e) Lin's (fuzzy set) solution [Lin90b]: There is a probability (or possibility) distribution on every aggregates. Applying that solution, some individual words can be released until the Counter reaches threshold.

Let us examine, how one can try to use these solutions to send a message in a network. Typically, the communication lines uses public phone lines, where the physical security is practically none. So one has to assume that a message between computers is available to the public.

1. If one uses solutions (a) or (d), no single words can be sent out.

2. If one uses solutions (c), few key words can not be sent out.

3. If one uses solutions (b) or (e), many words can be sent out until some threshhold is reached.

Any of these five solutions can not be used to send a message. However, we can accomplish the task with the notion of ordered aggregation; send the message with a random permutations.

One may argue that such communication can be done by encryption; we agree. Intrinsically, encryption is a means to hide information in the order; it is the same notion as ordered aggregation. However, we should like to point out that the notion of ordering has not been adequately discussed in the theory of cryptography. In traditional cryptography, the emphasis is on searching an "inverse algorithm" of an encryption, there is no systematic efforts to guard against a decryption by the **meaning** of data [Denn90]. To "destroy" the meaning of data, we need to permute the data randomly. This consideration had never been dicussed adequately in cryptography. Our research is overlapping with the "foundation" of cryptography, we will have reports on this issue in near future.

Example 2.2. If salary and name are both sorted in the same order, users can easily "guess" his coworkers salary. Hence the sensitive association will be compromised. To hide this senstive association, we need to randomly permute one of the attribute.

These examples, lead us to a serious consideration of the notion of randmness. Fortunately, there are many theory of randomness. Entropy or Kolmogorov randomness are most well known measurement of randomness. In view of Chaiten's results, for practical purpose, some modification is necessary. Via entropy, we propose the notion of quasi-Kolmogorov randomness.

As we have remarked at the introduction, there are many forms of entropy. Since the entropy in ergodic theory is closes to us, we will adopt the entropy from ergodic theory. In ergodic theory, the entropy is used to measure the randomness of a transformation of a space. By regarding a permutation of a message as a transformation of information, we can adopt the entropy in ergodic theory to measure the degree of "distortion" of a text. In fact, we can view the sentence structure (paragraph, sentence ...) as a partition of message, so we will use finite entropy, an analogue of entropy, to measure such a distortion (permutation). (See, e.g., [Perr69] for the notion of ergodic theory.)

Even though ergodic theory is not necessarily a popular subject in the community of computer security, we will not give an exposition on the ergodic theory. Instead, we will develop the notion of "finite entropy" in detail and occasionally make some comparison to the approaches in ergodic theory. Roughly, a "finite entropy" is a finite analogue of the entropy in ergodic theory. A real entropy of ergodic theory (living in infinite mathematic world) is always zero on any finite set; so for any reasonable finite applications, a modification is unavoidable. Roughly speaking, what we are going to do here is, basically, an adoption of the methodology in ergodic theory, but not (the direct applications of) its theorems.

3. PERMUTATION AND RANDOM PERMUTATION

In this section, we summarize some theory on permutations -- an elementary transformation group theory.

Definition. Let H be a group and Let

$$H \times S \longrightarrow S$$

be a map such that

(1) h(k(s)) = (hk)(s) for all s in S,
(2) Id(s) = s for all s in S,

Then such H is called a transformation group on S, or simply, H has an action on S.

Let S = {1,2,3,..., N} be a set of finite number. Let T be a permutation of S, i.e., a one-to-one map of S to S. Some special permutation can be represented by cycles: For example, let T be a permutation:

```
1 ----> 2
2 ----> 3
3 ----> 4
4 ----> 1
5 ----> 5
6 ----> 6
   :
N ----> N
```

The permutation T sends 1 to 2, 2 to 3, 3 to 4, then 4 is sent back to 1. Any numbers above 4 are fixed. Such a permutation is called a cycle and is denoted by (1234). The degree of (1234) is 4. These can be found in any standard abstract algebra book [Jacob56]. One can also find there the following theorem:

Proposition: Every permutation can be decomposed into cycles.

We will call such decomposition cycle decomposition.

Example: Let S = {1,2,3,4,5,6,7,8,9}, let the permutation T be:

```
1 ----> 7
2 ----> 5
3 ----> 8
4 ----> 2
5 ----> 4
6 ----> 6
7 ----> 1
8 ----> 3
9 ----> 9
```

Then T = (17)(254)(38)(6)(9). Normally, the cycle with one element is suppressed, namely, T = (17)(254)(38). Here (17) means 1 to 7 and 7 to 1. Similarly, (254) means 2 to 5, 5 to 4 and 4 to 2.

Definition: An orbit of H acting on S is an equivalence class of S with respect to the equivalence relation,

s is equivalent to r

iff there is an h in H such that h(s)=h(r).

By adopting, say Fortran's notation, we will denote nth power of

T, by T**n. The power of transformation can be defined inductively by

```
T**0 (S) = Id(S) = S,
T**n (S) = T(T**(n-1)(S)).
```

The set

```
T= {Id, T, T**2, T**3,....}
```

is a cyclic group and can be regarded as a transformation group on S. In this paper, we will use T either as a permutation or the cyclic transformation group. The actual meaning will be clear from the context. As an illustration, we will state the following proposition in our convention (abuse of notation).

Proposition. Let T be a permutation on S = {1,2,3,..n}. Then a subset A is an orbit space of T (as a group) on S if and only if A is a cycle in the cycle decomposition of T (as a permutation).

The proof is trivial, we will illustrate the notion by example. Let T =(17)(254)(38)(6)(9) and S ={1,2,3...9} as above. 6 and 9 above are the fixed points of T. {1,7}, {2,5,4} and {3,8} are orbit spaces of T. Orbit may also be called as minimal invariant subspace.

Definition. Let T be a transformation on S, then the map T, by abuse of notation

```
P(S) ----> P(S)
```

defined by

```
T({s1,s2,s3,..}) = {Ts1,Ts2,...}
```

is called diagonal action.

Proposition. If H is a transformation group on S, then, by diagonal action, H is a transformation group on the power set P(S). In particular, if T is a transformation on S, T is also a transformation on P(S).

Example. Let T=(17)(254)(38)(6)(9) and S ={1,2,3...9} as above. Then T send {2,4,8} to {5,2,3}. (T sends 2 to 5, 4 to 2 and 8 to 3).
Definition. A subset S1 of S is called i-subset if S1 contains i elements, i.e., the cardinality of S1 is i.

Proposition. Let P(S,i) be the family of all i-subsets of S. If H is a transformation group on S, then, by diagonal action, H is a transformation group on P(S,i). In particular, if T is a transformation on S, T is also a transformation on P(S,i).

Definition. The orbit which is not a fixed point in P(S,i) is called i-dimensional cycle.

Example. {1,3} is moved to {2,8}, and {2,8} is moved back to {1,3}. The family F={{1,3},{2,8}} is an orbit space of T on P(S,2) and hence is a 2-dimensional cycle.

Remark: To suppress the notation, in the future such family F will be simply denoted as 13,28.

4. FINITE ENTROPY

4.1 Partitions.

By a partition P = {Ri:i=1,2,..} of the set S, I mean a collection of pairwise disjoint sets Ri (i.e., the intersection of two distinct sets Ri and Rj is empty) and the union of all Ri is S. Ri will be called a component of the partition P.

Example: Let S = [a,b,c,d,e,f,g] be a set. Then

R1={a,c,e,g}, and
R2={b,d,f}

form a partition of S.

There are two trivial partitions, namely

(1) R1=S itself. The singleton P={R1} is a partition of S.
(2) Let P be a partition in which each component Ri contains only one element of S, i.e., R1={a}, R2={b} ...R7={g}, then P={Ri} is a partition.

We will call these two partitions improper partitions. A proper partition is a partition which is not improper. A partition P is said to be finer than another partition Q (or Q is coarser than P) if for every set A in P there is B in Q such that A is a subset of B.

Let P={Ri: i=1,2,..} and Q ={Sj: j=1,2,..} be two partitions. We denote by P+Q is the partition of S, which consists of all possible non-empty intersections of Ri and Sj. I sometimes simply say P+Q is the partition generated by P and Q. Such sum can obviously be generalized to more than two summands.
Let T be a permutation. T induces a new partition TP=P1 on S.

Example: Let S ={1,2,3,4,...n}. Let a partition be

P = { {1,3}, {2,4,6}. {5}, {7,8...n} }.

Let T be a cyclic permutation,

T(i) = i + 1 for i < n
T (n) = 1

(1--> 2, 2--> 3, ... (n-1)--> n, n-->1)
Then the partition TP=P1 is

P1 = {{2,4}, {3,5,7}, {6}, {8,9,...n ,1}}

Then, the next partition T**2 (P) = TP1 = P2 is

P2 = {{3.5}, {4,6,8}, {7}, {9,10,...,n,1,2}}

In general, T**i(P) = TP(i=1) = Pi is

Pi = {{i+1,i+3}, {i+2,i+4,i+6} {i+5}, {i+7,i+8,..n,1,2,..i}}.

Since S is a finite set, there can only be finitely many distinct partitions. So Pi will stop at some finite step.
Let the partition generated by all of the partitions P1,P2,..Pn be denoted by P[1,2,..n]. That is,

P[1,2,..n} = P1 + P2 + ...+ Pn.

We will denote by P[T] the partition generated by all possible different partitions P1,P2,..Pn,.. Note that since S is finite, there is at most finite number N of distinct partitions.
Thus,

P[T] = P1 + P2 + ... = P1 + P2 +...+ PN

Definition. A partition P is called a generator for a transformation T, if P[T]=P.

The following proposition follows immediately from the meaning of orbits.

Proposition. P is a generator if and only if P consists of orbits of T in various dimensions.

This proposition leads us to the final reduction theorem.

4.2 Finite Entropy.

Let P = {Ri:i-1,2,3,..} be the a partition. Let m be the counting measure, i.e., m(Ri) is the cardinal number of Ri. We define the entropy of P (as in Ergodic theory) as follow:

H(P)=[m(R1)*Log(m(R1)) + m(R2)*Log(m(R2)) + ...]

H(P) is called the entropy of the partition (* denotes the multiplication).

Next we define entropy of T and P by

H(P,T) = H(P[T])/Log|S|

where |S| is the cardinal number of S and P[T] is the partition on S generated by all the partitions of the powers of T.

We are interested in the entropy of a permutation, so we define

H(T) = AVE{ H(P,T): for all possible proper P in X}

where AVE, the average, is defined to be the SUM of H(P,T) divided by the number of all proper partitions P.

The entropy is defined in terms of the cyclic group generated by T. So any generator should have the same entropy. To choose among different generators, we offer the following: Since we are interested in ordering, we can assume the original space S is linearly order. Then for each permutation, we can speak of distance. We will define the distance of a permutation to be the maximum movements of the elements in S. The distance of an element moved is measured by the number of elements between the beginning position and ending position (after moved). We suggest choosing the generator which moves approximately half of the order of the permutation.

Remark:

In ergodic theory, the entropy H(P,T) is defined to be the limit of the sequence

{ 1/n * [H(P[1,2,..n]) }, as n ---> infinite

(* denotes the multiplication). The entropy of T then define as

H(T) = SUP H(P,T).

Roughly the entropy of T can be expressed as

H(T) = sup limit { 1/n*[H(P[1,2,...n])}
= sup limit {Cesaro's means}

In other words, roughly H(T) is the supremum of Cesaro means (Note that limit Ai = limit [(A1+A2...+An)/n]). We believe that our definition of H(T) is a finite analogue of the entropy in the ergodic theory.

4.3 Entropy of Cyclic Permutations and Identity.

(1) T=Id, the identity permutation.

Let P = {Ri:i=1,2,..} be any partition. The entropy is

H(P,T) = [m(R1)*Log(m(R1)+m(R2)*Log(M(R2)+.....]/Log|S|

If S has more than two points, I consider a partition, which consists of one point {s} and the rest of the points, S\{s}.

H(P,Id) = m(S\{s})*Log m(S\{s}) + m({s})*Log(m(S) > 0

So the entropy of Id is

H(Id) = AVE {m(S\{s}))*Log(m(S\{s})), ...} >0

(2) T is a cyclic permutation.

Let P be any partition. Then, because of the nature of cyclic permutation, P[T] is a partition of singleton, i.e., every equivalent class has one element. Thus

```
  H(T) = SUM[m(Ri)Log(m(Ri))/Log(m(S))]
       = SUM[(Log(r)]/Log(m(S))
       = 0 (in this case r=1)
```

The two extreme cases seem to be measuring what we expected.

5. COMPUTATION OF ENTROPY

The computation of entropy, via its definition, is prohibitedly expensive. The Hardy-Ramanuijn function HR(n) grows in the order EXP(Sqr(n)). And the number of partitions is strictly greater than HR(n) [HR(n) = the number of possible positive sum = n]. Therefore, without some reduction theorem, the computation is practically impossible. In this section, we introduced such a reduction theorem. The complexity of the reduction theorem is not depend on the input string n. It depends one structure of the representation of T as cycles.

5.1 Permutations with homogeneous cycles.

Let us examine a simple example:

S = {1,2,3,4,5,6} and T= (12)(34)(56).

Let A={1,2} be a component of some partition in S. Since A is invariant under T, it will be a component of the partition P[T]. Next, I would like to enumerate all the partitions that will give rise to a component A in P[T].

To properly enumerate the contribution of A to the entropy, we need some notations: Let C(n,m) be the binary coefficient, i.e., C(n,m) = n!/m!(n-m)!. Let Pi be the number of all possible partitions of i elements.

The possible partitions are:

(a) A2={1,2},and components of partitions of S\A2 = {3,4,5,6}
[for example, {1,2},{3},{4},{5},{6}
{1,2},{3,4},{5},{6} are such partitions]

(b) A3={1,2,-}, and components of all partitions of S\A3
[for example, {1,2,3},{4}.{5}.{6}
{1,2,4},{3}.{5}.{6} are such partitions]

(c) A4={1,2,-,-}, and components of all partitions of S\A4
[for example, {1,2,3,4},{5}.{6}
{1,2,3,4},{5.6}
{1,2,4,5},{3}.{6}
{1,2,3,4},{3.6} are such partitions]

(d) A5=(1,2,3,4,5} and components of all partitions ‹f {6}.

So the contribution of {1,2} to the entropy of T is:

E((12)) = (2Log2)[C(4,0)P4 + C(4,1)P3 + C(4,2)P2 + C(4,3)P1]
= 51(2Log2)

The terms C(4,0)P4 computes the case (a)
C(4,1)P3 computes the case (b)
C(4,2)P1 computes the case (c)

There are some over counting in this formula, however, there will be adjusted by the next formula

The contribution of {1,2,3,4} is:

E((12),(34)) = (4Log4 - 2Log2 - 2Log2)[C(2,0)P2 + C(2,1)P1]
= (4Log4-4Log2)*4

The two negative terms -2Log2 is to discount the over counting of the components of {1,2} and {3,4} at terms (c) and (d).

Combine all these formula together, we get:

E(T) = C(3,1)*(51*2Log2) + C(3,1)*(4(4Log4-4Log2))
= 129(2Log2) + 12(4Log4).

Besides cycles, there are other "higher dimensional orbit", namely, the orbits on P(S), the power set of S.

For example, {1,3}, {2,4} is such a 2-dimensional orbit.

The contribution of {1,3}, {2,4} can be understood from examining the possible partitions

The possible partitions are:

A2={{1,3}, {2,4}}, and components of all partitions of S\A2

[For example, {1,3},{2,4},{5},{6}
{1,3},{2,4},{5,6} are such partitions
the contribution to the entropy is (2(2Log2)*C(3,2)*2[c(2,0)P2
where the meaning of each term is:
2(2Log2) = m({1,3})Logm({1,3})+m({2,4})Logm({2,4})
C(3,2) = possible way of choosing two cycles
2 = possible orbits:{{1,3},{2,4}} and {{1,4},{2,3}}
c(2,0)P2 = contribution o: S\A2]

A3={{1,3,-}, {2,4}} and components of all partitions of S\A3
A4={{1,3}, {2,4,-}} and components of all partitions of S\A4
[for example, {1,3,5},{2,4},{6}
{1,3,6},{2,4},{5}
{1,3},{2,4,5},{6}
{1,3},{2,4,6},{5} are such examples]

A5={{1,3,-,-}, {2,4}} and components of all partitions of S\A5
A6={{1,3}, {2,4-,-}} and components of all partitions of S\A6
:
:
So the contributions of the 2-dimensional orbit are:

(2(2Log2)*C(3,2)*2[c(2,0)P2 + 2*C(2,1)P1 + 2*C(2,2)P0]
= 96(2Log2) - 8(3Log3)

Similarly, one can compute the 3-dimensional orbit

2(3Log3)*C(3,3)*(2**3)[C(0,0)P0]
So the entropy of T is:

H(T) = {129(2Log2) + 12(4Log4) + 96(2Log2) + *(3Log3)}/201

where 201 is the total number of partitions.

5.2 General Permutations.

First we observe that the major non-zero contribution to the entropy of a given permutation T comes from its orbits and its high dimensional analogue. The orbits of T are essentially some "collection" of cycles. The high dimensional orbits are "union" of cycles. The exact meaning will be clear from the proof.

Let T be a general permutation which is decomposed into cycles. These cycles are:

A1, A2, ..,Aa are cycles of degree Da
B1, B2,...,Bb are cycles of degree Db
C1, C2,...,Cc are cycles of degree Dc
:
:
and T is a product of these cycles. Let k=a+b+c+....
We will use E'(x) to represent the contribution of orbit x and E"(x) to represent the contribution of high dimensional orbits associated with x.

Let A be a typical cycle (i.e., one of those Ai', Bi', Ci',..) of degree D. By similar analysis of the example above, we can summarize the contribution of A as:

E'(A) = (dLogd){C(n-d,0)P(n-d) + C(n-d,1)P(n-d-1) +....+ C(n-d,n-d-1)P1}

Let B be another typical cycle with degree e, then the contribution of the orbits which contain A and B is:

```
E'(A,B) = ((d+e)Log(d+e) - dLogd - eLoge){C(n-d-e,0)P(n-d-e)
+C(n-d-e,d1)P(N-d-e-1) +...+C(n-d-e-1)P1}
```

Let C be another typical cycle with the same degree f, then the contribution of the invariant subspaces which contain A,B and C is:

```
E'(A,B,C) = { (d+e+f)Log(d+e+f) - dLogd - eLoge - fLogf
          -(d+e)Log(d+e) - fLogf - dLogd
          -(d+f)Log(d+f) - dLogd - eLoge)
          -(e+f)Log(e+f) - eLoge - fLogf) }
          * { C(n-d-e-f,0)P(n-d-e-f) + C(n-d-e-f,1)P(N-d-e-f-1)
              +...+C(n-d-e-f-1)P1 }
```

Similarly, one can find all the E' for 4, 5,..cycles

The contribution of high dimensional invariant subspaces can be computed as follows: Let A be one the Ai'.

```
E"(A) = (Pd - 1)*
             [take one component (point) of high dimensional
              cycle out, then take all possible partitions on the
              remaining collection of components]
{
d(2Log2)C(a,2)(d**(2-1)){c(n-2d,0)P(n-2d)
+d[C(n-2d,1)P(n-2-d-1) +...+C(n-2d,n-2d-1)P1+C(n-2d,n-2d)P0]}
                           [contribution of 2-dimensional cycles]

+d(3Log3)C(a,3)(d**(3-1)){C(n-3d,0)P(n-3d)+
d[C(n-3d,1)P(n-3d-1) +...+C(n-3d,n-3d-1)P1+3(n-3d,n-3d)P0]}
                            [contribution of 3-dimensional cycles]
          ........

+d(iLogi)C(a,i)(d**(i-1)){C(n-id,0)P(n-id) +
d[C(N-id,1)P(n-id-1) +...* C(n-id,n-id-1)P1+C(n-id,n-id)P0]}
                          [contribution of i-dimensional cycles;
                           i<=a, a is the number of A's]
}
```

Similarly, E"(B), E"(C),...
The final entropy is the weighted sum of E'(A), E'(A,B),.. and E"(A)... Formally, we stated

Reduction Theorem. Let T be a permutation on the set S. Then the entropy

```
H(T) = H'(T) + H"(T)
```

where

```
H'(T) = a*E(A1) + b*E(B1) + c*E(C1)+ ...
                                [sum of 1 representing cycles]

+C(a,2)*E(A1,A2)+C(a,1)*C(b,1)*E(A1,B1)+C(a,1)*C(c,1)*E(A1,C1)+..
                                 [sum of 2 representing cycles]

+C(a,3)*E(A1,A2,A3)+C(b,2)*C(c,1)*E(A1,A2,B1)
+C(a,1)*C(b,1)*C(c,1)*E(A1,B1,C1)+..
                                 [sum of 3 representing cycles]

+[C(a,k1)*C(b,k2)*.....]*E(A2,...B1,B2,..C1,C2.....)
+.......
                             [sum of k-1 representing cycles;
                              k is the total number of cycles;
                              k1+k2+... = k-1]
H"(T) =a*E"(A1)+b*E"(B1)+c*E"(C1)+ ...
                              [sum of representing cycles]
```

With this theorem, the computation of entropy of a transformation T becomes a counting problem. Using the reduction theorem the computing time is drastically reduced. The complexity of this computation is not depends on the number of input, but depends on the structure of input; the structure of cyclic decomposition of the permuatition.

6. QUASI KOLMOGOROV RANDOMNESS

Suppose we are given a binary Kolmogorov random string. Then by appropriate permutation, the string can be reduced to 0's and 1'a -- we will call such string commutative Kolmogorov string. We have a good knowledge about the number of 0 and 1's in such a commutative Kolmogorov random string. Now we use an entropy random permutation to permute such a commutative Kolmogorov random binary string. Then the resulting string should be a good approximation to the Kolmogorov random string. We will call such string quasi Kolmogorv random string. The permutation which bring a Kolmogorov random string to the commutative string will be called Kolmogorv permutation. We believe entropy random permutation is an approximation of Kolmogorov permutation. Further will be conducted.

7. CONCLUSION

Most of physical states are random, in other words, the number of random states should dominate the number of ordered states. Similarly the number of random strings should be much more than that of ordered strings. In fact, this is one of our intuitive assumption on the properties of randomness. If there is, say in the extreme case, only one random string, then we can always looking for clues from this "random string"; and the meaning of randomness is lost.

Any notion of "computable randomness" can not coincide with Kolmogorov randomness; otherwise by Chaiten's theory, it will imply that the "computable randomness" is not computable. Conversely, for any notion of "computable randomness", there are strings which are "computable random", but not Kolmogorov random. Therefore, some of the string which is "entropy random", may not be Kolmogorov random. The cyclic permutation has a low entropy, but intuitively, it can not be Kolmogorov random.

The computation of entropy is quite interesting, it involves (1) Ergodic theory (2) transformation group theory in a very innovative fashion (No transformation group theory have looked at high dimensional diagonal action and investigate their orbits (the high dimensional cycles); this may create new field in transformation group theory. Though the reduction theorem reduces the computation in the order of magnitude, it still quite complicated, some notion of approximate entropy may be needed in order to reduce the computation further.

Acknowledgement

The second author was partially supported by NSA(MDA904-91-C-7048), and RADC. He would like to thank Howard Stainer and Michael Ware, and Joseph Giodano for their supports.

REFERENCES

[Ashby91] V. Ashby, "Practical Cardinality Aggregation Problem", Proceedings of 4th RADC Database Security Workshop, 1991.

[Chai87b] G.J. Chaiten. Algorithmic Information Theory, Cambridge University Press, 1987.

[Denn86b] D.E. Denning, S.G. Akl, M. Heckman, T.F. Lunt, M. Morgenstern, P.G. Neumann, and R.R. Schnell. "View for Multilevel Database Security," Proceedings of IEEE Symposium on Security and Privacy, 1986.

[Denn87a] D.E. Denning, T.F. Lunt, R.R. Schell, M. Heckman, and W.R. Shockley, "A Multilevel Relational Data Model," Proceedings of 1987 IEEE Symposium on Security and Privacy, 1987.

[Denn87b] D.E. Denning, T.F. Lunt, R.R. Schell, M. Heckman, and W.R. Shockley. "The SeaView Formal Security Policy Model," Computer Science Laboratory, SRI International, July 1987.

[Denn88a] D.E. Denning, T.F. Lunt, R.R. Schell, M. Heckman, and W.R. Shockley. "The SeaView Security Model," Proc. 1988 IEEE Symposium on Security and Privacy, 1988.

[Denn88b] D.E. Denning, T.F. Lunt, P.G. Neumann, R.R. Schell, M. Heckman, and W.R. Shockley. "Security Policy and Policy

Interpretation for a Class A1 Multilevel Secure Relational Database System," Computer Science Laboratory, SRI International, Aug. 1988.

[Dill87] B.B. Dillaway et al. "Security Policy Extensions for a Database Management System." Interim Report A002, Honeywell Systems Research Center and Corporate Systems Development Division, April 1987.

[Dill88] Dillaway et al. "Security Policy Extensions for a Database Management System." Interim Report A002, Honeywell Systems Research Center and Corporate Systems Development Division, May 1988.

[Dwye87] P.A. Dwyer, G.D. Gelatis, and B.M. Thuraisingham. "Multi-level Security in Database Management Systems," Computers & Security 6 (1987), Elsevier Science Publishers B.V. (North Holland), pp. 252-260.

[Dwye88] P. Dwyer, E. Onuegbe, P. Stachour, and B. Thuraisingham. "Secure Distributed Data Views_Implementation Specifications." Interim Report A005, Honeywell Systems Research Center and Corporate Systems Development Division, May 1988.

[Garvey91] T. Garvey, T. Lunt and M. Stickel, Characterizing and Reasoning about Inference Channels", Proceed. of 4th RADC Database Security Workshop, April 1991

[Garvey91] T. Garvey, T. Lunt and M. Stickel, Abductive and Approximate Reasoning Models for Characterizing Inference Channels", Proceed. of 4th Workshop on the Foundations of Computer Security, Franconia, New Hampshire, June 1991

[Hink88] T.H. Hinke. Inference aggregation detection in database management systems. In Proceedings of the 1988 IEEE Symposium on Security and Privacy, April 1988.

[Hink88] T.H. Hinke, C. Garvey, and A. Wu. A1 secure DBMS architecture. In Research Directions in Database Security, T.F. Lunt, editor, forthcoming.

[Jacob56] N. Jacobson, Abstract Algebra, van Nostrand, 1956

[Jens88] N. Jensen. "System Security Officer Functions in the A1 Secure DBMS," Proc. of the 1988 Workshop on Database Security, Oct. 1988.

[Land81] C.E. Landwehr. "Formal Models for Computer Security," Computer Surveys," Vol. 13, No. 3, September 1981.

[Land82] C.E. Landwehr and C.L. Heitmeyer. "Military Message Systems: Requirements and Security Model," NRL Memorandum Report 4925, Computer Science and Systems Branch, Naval Research Laboratory, Sept. 1982.

[Lin89a] T.Y. Lin. "A Generalized Information Flow Model and Role of System Security Officer," Database Security: Status and Prospects, Vol. 2, C.E. Landwehr, ed. North Holland, 1988.

[Lin89b] T.Y. Lin, L. Kerschberg, and R. Trueblood. "Security Algebra and Formal Models." Proceedings of IFIP WG11.3 Workshop on Database Security, September 5-7, 1989.

[Lin89c] T.Y. Lin. "Commutative Security Algebra and Aggregation. "Proceedings of Second RADC Workshop on Database Security, 1989.

[Lin90a] T.Y. Lin, Database, Aggregations and Security Algebra," Proceedings of IFIP WG11.3 Workshop on Database Security, September 18-21, 1990.

[Lin90b] T.Y. Lin. Probabilistic Measures on Aggregations, Proceeding of 6th annual Application Computer Security Conference,Dec. 3-7, 1990.

[Lin90c] T.Y. Lin. Message and Non Commutative Aggregation, Proceedings of 3rd RADC Database Security Workshop, June 1990

[Lin90d] T.Y. Lin. Multlevel Database and Univesal Security Algebra - A Prelimnary Report, Proceedings of 3rd RADC Database Security Workshop, June 1990

[Lin91a] T.Y. Lin. "Multilevel Database and Aggregated Security Algebra." Database Security: Status and Prospect III, 1991, a revision of the paper in the Proceedings of IFIP WG11.3 Workshop on Database Security, September 18-21, 1990.

[Lin91b] T.Y. Lin. "Inference" Free Multilevel Database Systems, Proceed. of 4th RADC Database Security Workshop, April 1991

[Lin91c] T. Y. Lin "Ordered Aggregation, Association and Encryption", Proceed. of 4th RADC Database Security Workshop, April 1991

[Lipn82] S.B. Lipner. "Non-Discretionary Controls for Commercial Applications, " Proc. 1982 IEEE Symposium on Security and Privacy, 1982.

[LiVi88] Ming Li and Paul Vitdnyi. "Two Decades of Applied Kolmogorov Complexity," Proceedings of Third IEEE Structure in Complexity Theory Conference, 1988.

[Lunt83] T.F. Lunt, D.E. Denning, P.G. Neumann, R.R. Schell, M. Heckman, and W.R. Shockley. "Final Report Vol. 1: Security Policy and Policy Interpretation for a Class A1 Multilevel Secure Relational Database System." Technical Report, Computer Science Laboratory, SRI International, Menlo Park, CA, 1988.

[Lunt88a] T.F. Lunt, R.A. Whitehurst. "The SeaView Formal Top Level Specifications," Computer Science Laboratory, SRI

International, Feb. 1988.]

[Lunt88b] T.F. Lunt, D.E. Denning, R.R. Schell, M. Heckman and W.R. Shockley. "Element-Level Classification with A1 Assurance," Computers & Security, Vol. 7, No. 1, 1988, pp. 73-82.

[Lunt88c] T.F. Lunt, R.R. Schell, W.R. Shockley, M. Heckman, D. Warren. "A Near-Term Design for the SeaView Multilevel Database System," Proceedings of 1988 IEEE Symposium on Security and Privacy, 1988.

[Lunt88d] T.F. Lunt, R.A. Whitehurst. "The SeaView Formal Top Level Specifications," Computer Science Laboratory, SRI International, Feb. 1988.

[Lunt88e] T.F. Lunt. "Access Control Policies: Some Unanswered Questions," Computer Science Laboratory, SRI International, June 1988.

[Lunt88f] T.F. Lunt. "Multilevel Database Systems: Meeting Class A1," Proceedings of the 1988 Workshop on Database Security, Oct. 1988.

[Lunt88g] T.F. Lunt, D.E. Denning, R.R. Schell, M. Heckman and W.R. Shockley. "Element-Level Classification with A1 Assurance," Computers & Security, Vol. 7, No. 1, 1988, pp. 73-82.

[Lunt88h] T.F. Lunt, R.R. Schell, W.R. Shockley, M. Heckman, D. Warren. "A Near-Term Design for the SeaView Multilevel Database System," Proc. 1988 IEEE Symposium on Security and Privacy, 1988.

[Lunt89a] T.F. Lunt. "Aggregation and Interference: Facts and Fallacies." Proceedings of the 1989 IEEE Symposium on Security and Privacy, Oakland, CA, May 1989.

[Lunt89b] T.F. Lunt, D.E. Denning, R.R. Schell, M. Heckmann, and W.R. Shockley. "Final Report Vol. 2: The Sea View Formal Security Policy Model." Technical Report, Computer Science Laboratory, SRI International, Menlo Park, CA 1989.

[Lunt89c] T.F. Lunt. "Real-Time Intrusion Detection," Proc. COMPCON Spring 1989.

[Mchu86]. "An EMACS Based Downgrader for the SAT," Computer and Network Security, M.D. Abrams and H.J. Podell, eds., 1986.

[Mill76] J.K. Millen. "Security Kernel Validation in Practice," Communications of the ACM, Vol. 19, No. 5, May 1976, pp. 243-250.

[Morg86] D.E. Denning and M. Morgenstern. "Military Database Technology Study: AI Techniques for Security and Reliability. Technical Report, Computer Science Laboratory, SRI International, Menlo Park, CA 1986.

[Morg87] Mathew Morgenstern. "Security and Interference in Multilevel Database and Knowledge-base Systems," ACM International Conference on Management of Data (SIGMOD-87), May 1987.

[Morg88] Mathew Morgenstern. "Controlling Logical Inference in Multilevel Database Systems, Proceedings of 1988 IEEE Symposium on Security and Privacy, 1988.

[Ostm56] H. Ostmann. Additive Zahkentheorie, Springer-Verlag, 1956.

[Parr69] W. Parry, Entropy and Generators in Ergodic Theory. Benjamin, 1969.

[Poly68] G. Polya, Patterns of Plausible Reasoning, Princeton University Press.

[Sayd87] O.S. Saydjari, J.M. Beckman and J.R. Leaman, "Locking Computers Securely," Proc. 10th National Computer Security Conference, Sept. 1987, National Bureau of Standards, National Computer Security Center, pp. 129-141, 1987. Advances in Computer System Security, Rein, Turn, ed., Vol. 3, pp. 207-219, 1988.

[Suoz87] Tzong-An Su and Gultekin Ozsoyoglu. "Data Dependencies and Inference Control in Multilevel Relational Database Systems," IEEE Symposium on Security and Privacy, Oakland, CA, April 1987.

[Tayl84] T. Taylor. "Comparison Paper Between the Bel and LaPadula Model and the SRI Model," Proc. 1984 IEEE Symposium on Security and Privacy, 1984.

[Thur87] M.B. Thuraisingham. "Security checking in relational database management systems augmented with inference engines." Computers and Security, Vol. 6, No. 6, 1987.

[Thur90a] M.B. Thuraisingham. "The Inference Problem in Database Security," CIPHER Newsletter, 1990.

[Thur90b] M.B. Thuraisingham. "Novel Approaches to Handling the Inference Problem in Database Security," Proc at 3rd RADC Database Security Workshop, Castile, 1990.

SECTION 3

POLYINSTANTIATION AND INTEGRITY

Polyinstantiation continues to be a controversial issue for multilevel database management systems. However, it is only one of several problems that result from conflicts between integrity features of database management systems and the enforcement of mandatory access controls. In their paper "Polyinstantiation in Relational Databases - Some Semantic Questions," Yazdanian and Eizenberg identify fundamental ambiguities that result from polyinstantiation and suggest alternatives for introducing appropriate semantics.
Bill Maimone and Richard Allen describe a number of areas where multilevel security enforcement conflicts with the integrity features defined by the ANSI SQL2 standard. Their paper "Methods for Resolving the Security vs. Integrity Conflict" provides insight into the problems that face both application designers and DBMS vendors when attempting to provide multilevel security in a database environment. Chip Paradise's paper "Using Polyinstantiation in the Real World" describes a government effort to employ polyinstantiation to provide "cover stories" in a multilevel database application. In their paper "Enforcing Primary Key Requirements in Multilevel Relations," Sushil Jajodia and Ravi Sandhu propose several alternatives for providing integrity without sacrificing strict security. Leonard Binns' paper "Inference Through Polyinstantiation" analyzes some potential inference exposures in applications that intentionally use polyinstantiation to provide cover stories.

POLYINSTANTIATION IN RELATIONAL DATABASE - SOME SEMANTIC QUESTIONS

G. Eizenberg - K. Yazdanian

ONERA/CERT - DERI
2 Avenue Edouard Belin
31055- Toulouse cedex
France
fax (+33) 61 55 71 72
tel:
(+33) 61 55 70 64 - (+33) 61 55 70 75
e-mail:
eizenber@tls-cs.cert.fr - yaz@tls-cs.cert.fr

Abstract

In a Multilevel Relational Database, security problems occur when data are handled at different protection levels. Semantic ambiguities which result from the polyinstantiation solution are pointed out and some clarifications based upon the interpretation of the key unicity are proposed.

In a relational database, facts are represented through tuples of a relation. In the basic flat relational model any two different tuples correspond to different information but introducing the concept of key and functional dependencies between relation attributes allows to identify an entity with its key attributes values. Thus two different tuples with the same key value, e.g. in two different database states, may be interpreted as two different values of the same entity.

If the key unicity constraint is enforced in any relation extension, each key value is present at least once. Therefore it is possible to define the update operation as modifying non key attributes of a tuple identified by its key attributes. Modifying one or more key attributes is not considered as an update operation but as a couple of delete and insert operations.

The key unicity assumption meaning may be recalled: every non key attributes should depend on the key.

Without the key unicity assumption, it is possible to delete and insert tuples but not to update entities in a single operation. Moreover, if the key does not include all the relation attributes, a relation extension may contain different instances of the "same thing". This is called polyinstantiation. Logically forbidden in a classical database instance, its necessity has been claimed in the multilevel security database in order to prevent information leakage from high level to

low level by covert channels [Den&al]. The polyinstantiation is handled by integrating security labels as new attributes into the relational model.

Several interpretations of the polyinstantiation and their consequences will be shown.

ABSOLUTE OR RELATIVE TRUTH

Several users of different clearances are using (querying, updating) the same database (relation) where several data (tuples) are diversely classified.

A leakage may occur when a low level user tries to insert a tuple whose key value already exists at a higher level. If no security management is done, the database management system logically rejects the new insert. Hence, the low user is aware of the existence of the data at a higher level, since querying the database about his own data gives no answer. A potential signalling channel results: a high level subject inserts and deletes a tuple with a given key, while a low subject tests the effect of the insertion of a tuple having the same key value. To avoid this kind of leakage, an "entity polyinstantiation" has been proposed [LuHs1].

The existence of a tuple (i.e. its key value) may have a low level while its "value" (i.e. its non key attributes values) is highly classified. Such a situation results from the following sequence of operations:

1. a low subject creates a low tuple (the non key attributes have 'null' or defined values)
2. a high subject updates some of the non key attributes of the same tuple (the previous instance of the tuple does not exist any longer)

Now, if a low query involves this high level tuple, it is prohibited, this reveals the previous high level update. In order to avoid the leakage which may result from that sequence, "polyinstantiation" consists in accepting two tuples sharing the same key values: the initial one and the updated one. The next example is proposed in [JaSa]:

The multilevel relation SOD (Starship, Objective, Destination) has a user primary key Starship to say that a starship has one objective and one destination only. This is true for a monolevel usage of the database, but with a multilevel security policy, it is meaningful to have some classified data on the starship "Enterprise" which are not visible for low level users. In this case a low level user could or would introduce an objective and a destination for the "Enterprise".

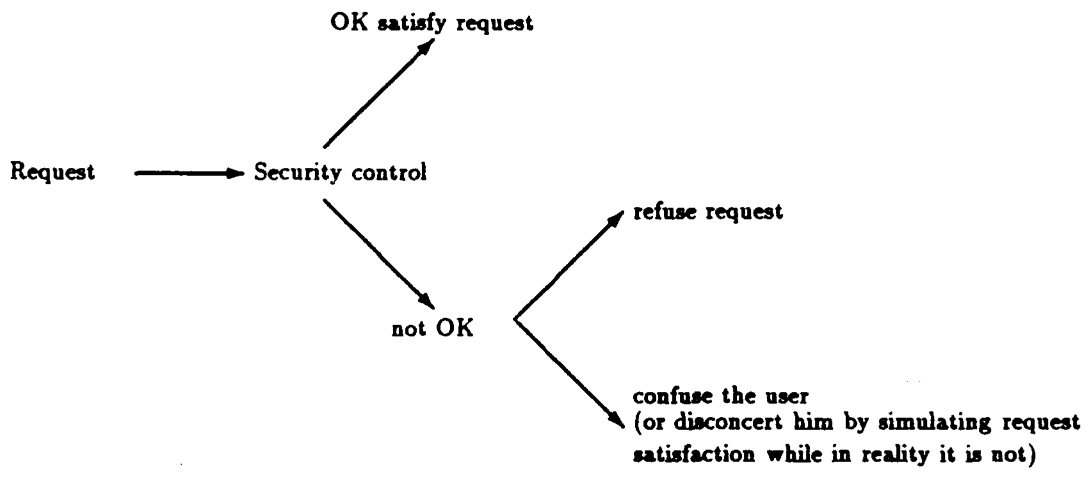
One way to avoid polyinstantiation would be to create a database extension for each of the security levels. Hence, the low level behavior of the database would be the same as if no higher level data existed. Some functionalities would not exist any longer. For instance, it would not be possible to use the security attributes as selection arguments. But when and why is it unacceptable? It would be helpful to make the rationales more explicit.

In fact, a high clearance user must continue to "see" any data whose level is dominated by its own clearance level and therefore will see two data with the same key (polyinstantiation). If we consider the semantic of the primary key, this is not allowed when dealing with facts in the real world, apart of any multilevel consideration. In the above example, "Enterprise" as any other starship could not have more than one objective and destination. Notice that if we admit that the "Enterprise" could really have two simultaneous missions - one secret and the other unclassified - the Starship attribute would NOT be a key of the relation. The question is: what kind of semantic for the two sibling tuples or what kind of acceptable constraint could solve the non unique key value problem?

In the multilevel mode, two basic choices are possible :

- Telling the truth to any user
- Lying to low level users

These two basic choices are general in any security system management :


OK satisfy request
Request
Security control
refuse request
not OK
confuse the user
(or disconcert him by simulating request satisfaction while in reality it is not)

In the "telling the truth" option, the low level user trying to introduce another objective and destination for the "Enterprise" is aware that such a tuple already exists at a high level he is not allowed to access. Therefore he is not allowed to introduce another objective and destination for this starship due to the general consistency rule "A starship has only one objective and one destination" which is expressed through the user primary key of the relation SOD. We have seen that a leakage may result.

In the "Lying option", the low level user is not aware of the existence of a high tuple (although it really exists) and is allowed to introduce another one of its own level. Of course, considering the general rule "A starship has only one objective and one destination" the new tuple introduced by the lower level user looks inconsistent with the old one. Is it really inconsistent?. This tackles the problem of integrity while dealing with a multilevel relation. In this case a high level user will see both tuples and must be aware that at most one of them is true. Let us assume the lower level tuple is not true. It may be on purpose (a lure) or erroneous: this ambiguity cannot be easily clarified.

There is another problem about the integrity of data. Any user whose clearance is not maximal has no guaranty on the data integrity. His only assurance when obtaining answers to his interrogations is: "these are the facts I am supposed to believe". Is it acceptable? Is this always admissible for a multilevel database where reliable data must exist at any level?

If one does not accept neither the consequences of the lying option nor potential covert channels, a solution is to consider reality (facts) as being relative to the security level linked to it. Hence, the user primary key unicity is applied to each level. Otherly stated, the classification level is part of the primary key, and the semantic of the relation is different: there are not true data in one hand and false data in the other hand but relatively true data for each level and low level users are free to introduce "their" true data for "their" level. Of course, for a given level (same classification level attribute value), due to the general rule "A starship has only one objective and one destination" the primary key is unique (no more than one tuple for the "Enterprise" starship).

In this case, high level users will "see" low level data associated with their classification level and are naturally aware of the fact: this data is true for that level.

The relative truth approach makes it necessary to answer to the following semantic question: what are the 'Low' informations that are considered as being true by the 'High' users? For instance, the tuple

SOD(Enterprise, Fishing, Grenade)

is created at the Low level. Clearly, if the High user updates an attribute (e.g. the Objective becomes 'War'), he does not believe the previous information any longer. But did he already believe it before the update operation?

Another kind of question comes from the polyinstantiation if the "Lying option" is chosen. Suppose that both next tuples exist in an instance of the database :

SOD(Enterprise [U], Fishing [U], Grenade [U])

SOD(Enterprise [U], War[S], Grenade [U])

'U' and 'S' stand for Unclassified and Secret respectively.

The polyinstantiation technique implicitly accepts the following fact:

"The U user is not authorized to know that he is not authorized to know the Objective of the Enterprise"

This fact is a direct consequence of the dynamic binding of the security levels (in other words, the update operation has been done at the current level S). But another choice could be made. If one accepts that :

"U is authorized to know that he is not authorized to know the Objective of the Enterprise"

then it can be stated, at the level U, that the Objective level of the Enterprise is S. In such a case, the lure and the polyinstantiation are not necessary any longer.

To summarize, at each level there is only one tuple occurrence per user key value, which means that including the tuple protection level in the primary key will ensure the "one tuple per relation key value" rule of the database.

OTHER PROBLEMS

The approach presented above concerns the problem of polyinstantiation for relations where the primary key is not the whole set of its arguments. There are other problems when either the key is the whole set of the attributes or there are several candidate keys.

The key is the whole set of attributes

For instance R(P,E,L) describes a (project,experiment,laboratory) relation, with different experiments taking place in different laboratories for different projects. In realistic situations, the primary key is the whole set (P,E,L): no one attribute is functionally depending on the two others. In such a case, all the attributes of a tuple have the same security level which is the level of the tuple and the knowledge of the existence of the data is equivalent to the knowledge of the data itself. The previously proposed semantic looks the most appropriate to this case: the truth is relative to the security level and a low level clearance user who attempts to insert a data a high classified occurence of which already exists must not be warned.

Several keys

Let us consider a multilevel relation with two candidate keys. Any of the keys can identify the existence of a tuple. If both keys have the same security level in any tuple, the conclusions and questions of the single key case analysis remain valid. The new case to consider is: two keys may have different levels in some tuples.

For instance let us consider the relation

E(Empno, Name, Age, Function)

for employee number, age, name and function respectively.

It is assumed that there are two key candidates: Empno and the couple (Name, Age)

Suppose the next tuples are in an instance of the database:

('id704' [U], (Smith, 37) [U], Programmer [U]) ('id323' [U], (Smith, 37) [S], Manager [U])

('id323' [U], (Jones, 29) [U], Manager [U])

Now the employee (Smith, 37) has left the company. The database must be updated. A U level user requests to delete the 'id704' tuples. Consequently, the first tuple is deleted. But equivalently for him, he may request to delete the (Smith, 37) tuples. Then the two first tuples will be deleted. Thus, the semantics of the two operations are not the same whereas the intention of the user is the same.

It seems that some further constraints must enforce uniform classifications for all keys (candidates or not) in order to be able to check multiple key consistency at any level.

CONCLUSION

The polyinstantiation is a technical solution to security problems in Multilevel relational databases, but the corresponding semantic is ambiguous or questionable unless a priori assumptions are explicitely made. Depending on the assumptions, there may be severe consequences on the integrity. A promising way to get rid of ambiguities while accepting more diverse semantics will be to state the facts and constraints in modal first order logic and then considering how a given database or knowledge base can be used to implement them.

BIBLIOGRAPHY

[Den&al] D.E. DENNING, T.F. LUNT, R.R. SCHELL, W.R. SHOCKLEY, M. HECKMAN: "The SeaView Security Model" IEEE Symposium on Security and Privacy - Oakland (1988)

[DwyJelThu] P. A. DWYER, G.D. JELATIS, B.M. THURAISINGHAM: "Multilevel Security in Database Management Systems" Computer & Security 6 (1987)

[GaLu] T.D. GARVEY, T. F. LUNT: "Multilevel Security for Knowledge Based Systems" Sixth Computer Security Applications Conference - Tucson, Arizona, Dec 1990

[JaSa] S. JAJODIA, R. SANDHU: "Polyinstantiation Integrity in Multilevel Relations" IEEE Symposium on Security and Privacy - Oakland (1990)

[LuHs1] T. F. LUNT, D. HSIEH: "Update Semantics for a Multilevel Relational Database System" Fourth IFIP WG11.3 Workshop on Database Security - Halifax, England, Sept 1990

[LuHs2] T. F. LUNT, D. HSIEH: "The SeaView Secure Database System: A Progress Report" ESORICS 90 - Toulouse, France, Oct 90

Methods for Resolving the Security vs. Integrity Conflict

Bill Maimone & Richard Allen
Oracle Corporation
400 Oracle Parkway, 14th Floor
Redwood Shores, CA 94065

INTRODUCTION

This paper considers the conflicts between security and integrity in a multilevel secure (MLS) RDBMS, including the entity and referential integrity conflicts described in [1], plus value constraints and transaction integrity. In each of these conflicts, several alternatives for resolving the conflict are described which result in different tradeoffs in the degrees of security and integrity. Additional consideration is given to SQL features specified in ANSI SQL [2,3] which introduce new security and integrity issues.

The security policy considered here follows the common mandatory access control (MAC) policy described in the Trusted Computer System Evaluation Criteria [4]. A subject can read a data element only if the subject access class dominates the object access class, and can write a data element only if the object access class dominates the subject access class.

ENTITY INTEGRITY

Entity integrity requires that any tuple can be uniquely identified by its primary key. More specifically, each tuple contains a non-null primary key whose value is different from all other values in that relation. The following is an example of the standard ANSI SQL syntax used to create a table.

```
CREATE TABLE DEPT
        (DEPTNO NUMBER PRIMARY KEY,
        DNAME CHAR(10) NOT NULL)
```

The security conflict occurs when a user at a low access class attempts to insert a primary key value which has already been inserted at a high access class. If the RDBMS does not insert the row, the secrecy of the high value is compromised. If the RDBMS inserts the row, entity integrity is compromised.

One approach to strictly maintain secrecy in spite of integrity is called polyinstantiation. This approach effectively considers the access class to be an "invisible" part of the primary key. Duplicate primary key values are permitted, provided that no two duplicates are inserted at the same access class. Numerous papers [5,6,7] have explored the consequences of entity integrity under polyinstantiation. The undesirable effects of polyinstantiation are greatly reduced where the unit of polyinstantiation is reduced to the single element level, but any polyinstantiation scheme faces additional complexity in a relational database which does not support multi-valued attributes.

Another approach which may maintain secrecy, but without sacrificing entity integrity, is to avoid possible duplicate key value conflicts by restricting a user's choice of primary key values. This might be accomplished either through a controlled application environment, or through a DBMS-supplied unique key value. For many applications the user is not relied upon to supply a unique primary key value, simply because it is operationally inconvenient to have a user guess a unique value. Well-written applications will automatically find a new unique value, typically by reading and incrementing a sequence number. This "convenience" might be strengthened by ensuring that all new records are inserted using a controlled application, or by requiring the value to be supplied by the DBMS. The security exposure of this approach lies in the generation of unique values. If users at multiple access classes read and increment the same sequencer, there is a well-known covert channel with a bandwidth proportional to the speed of the sequencer. The sequencer channel can be reduced by inserting delays or by introducing a random element into the sequence generation. The channel can be eliminated by partitioning the sequence values according to access class. This latter method on the surface seems equivalent to polyinstantiation, but it is significantly different in that the primary key maintains uniqueness without considering an "invisible" access class. The sequencer approach is not helpful where the user must in fact choose the primary key to match some existing real world value.

Finally, one might choose to simply notify the user of the duplicate key value and compromise security. This approach may be acceptable in conjunction with a specific auditing capability and/or automatic audit actions to minimize the compromise. For example, a duplicate value error might automatically revoke the user's access or notify a security officer.

Since some of these approaches could be specific to particular relations or transactions, it may be appropriate to vary the method used within an application according to such factors as the degree of user access allowed, frequency of change, actual data sensitivity, and auditing capabilities.

REFERENTIAL INTEGRITY

The referential integrity property states that a foreign key value must always refer to a valid primary key value. In other words, there must be no dangling references. A security conflict arises whenever there is a foreign key relationship across two access classes. It can be assumed that the foreign key (child) record must always dominate the primary key (parent) record; otherwise a user reading the child record would observe an apparent integrity violation. The conflict occurs when a parent record is removed after a child record has been created at a higher access class. A user removing a parent record may be incapable of removing or even detecting all dependent child records at higher access classes. The following is an example of ANSI SQL syntax for creating a dependent child table called EMP with a foreign key pointing to the previously created DEPT table.

```
CREATE TABLE EMP
        (EMPNO NUMBER PRIMARY KEY,
        ENAME CHAR(20) NOT NULL,
        SALARY NUMBER(7,2)
        CHECK (SALARY > 0),
        DEPTNO NUMBER REFERENCES DEPT (DEPTNO))
```

One approach is to allow the parent record to be deleted, and to leave the child records in place. This ensures secrecy, but sacrifices referential integrity since the child records now refer to a non-existent parent. One method of resolving the inconsistency is to remove the child records at some later time with a garbage collection procedure. The integrity risk is that the dangling child records might be seen before they are deleted and that some application will interpret this as a valid record. Even worse, the orphan record might accidentally be "adopted" if the previous parent primary key value is reused before the child records are removed. To eliminate this risk completely, each application would be required to incorporate the garbage collection procedure into each operation. If the security model prohibits writing up, garbage collection would have to be performed on both read and write operations.

Another approach is to remove or redefine the referential constraint in such a way that it does not violate security. The ANSI SQL2 referential actions "cascade", "set null", and "set default" all provide a means for dealing with classified child records in a more secure manner [3]. The SQL column syntax for "cascade" is shown below; the other two options are similar.

```
DEPTNO NUMBER REFERENCES DEPT (DEPTNO)
        ON DELETE CASCADE
```

The cascade option automatically removes the child records as part of the same transaction as the removal of the parent – equivalent to immediate garbage collection but without any intermediate adoption problems. The set default/null options leave the child records in place, but change the foreign key value so that it refers either to another valid value, or to null. The latter may require a change to the data model; the relationship must be optional. From another perspective, these referential actions are simply the garbage collection mechanism required above. The difference is that for MLS systems the mechanism requires the ability to write up in order to enforce the constraints. The resulting security risk is that the user might be able to directly or indirectly learn the nature or existence of the write up operation.

The ANSI "restrict" referential action is the one option which does not preserve security. This option would prevent the user from deleting the parent until all dependent child records were deleted. If the user cannot see the offending child records, he can infer the existence of child records at a higher access class. As with entity integrity, the exposure of this approach can be reduced by restricting record deletion, or by auditing the enforcement of the restriction.

For any of the above referential constraint enforcement mechanisms which write above the user's access class, there is a non-obvious complexity introduced by the trigger mechanism described in SQL3 [3]. In brief, a trigger causes some actions to be executed before or after some predefined event, such as record insertion or deletion. For example, the following trigger might record salary changes.

```
CREATE TRIGGER SALARY_RECORD
        BEFORE UPDATE OF SALARY ON EMP
        FOR EACH ROW
BEGIN
        INSERT INTO SALARY_HISTORY
        (EMPNO,OLD_SALARY,NEW_SALARY)
        VALUES (OLD.empno,OLD.salary,NEW.salary);
END;
```

The difficulty arises when triggers are combined with the three "secure" referential actions. In particular, the write-up carried out to enforce referential integrity may cause a trigger to fire. Since the trigger is firing as a result of the existence of data above the access class of the user, the direct or indirect results of the trigger should not be available to the user. Direct trigger results such as rows inserted into another relation can be correctly labeled at the access class of the triggering event or data. However extreme caution must be exercised whenever writeup is required to perform an operation. All of these actions may be used to cause code to execute at a higher access class. Since the semantics of triggers allows a trigger to cause a transaction to fail, trigger code may be designed to effect the transaction commit or rollback depending on the contents of classified data. For example, the following trigger will tell an unclassified user whether or not the sensitive employee Allen's salary is greater than $100,000. The trigger fires when an unclassified user attempts to delete Allen's department.

```
CREATE TRIGGER SALARY_TRACKER
        BEFORE DELETE ON EMP
        FOR EACH ROW
DECLARE
        high_salary EXCEPTION;
BEGIN
        IF OLD.salary>100000 and OLD.ename = 'ALLEN'
        THEN RAISE high_salary;
END;
```

Finally, an approach which avoids the problem is to restrict references so that the parent and child must always be at the same access class. This would prevent any possible channels, since those channels only exist where a reference crosses multiple access classes. This mechanism can be implemented using the following trigger on EMP and a similar one on DEPT.

```
CREATE TRIGGER AVOID_EMP
        BEFORE INSERT ON EMP
        FOR EACH ROW
DECLARE
        deptlabel LABEL;
        label_mismatch EXCEPTION;
BEGIN
        SELECT ROWLABEL FROM DEPT INTO deptlabel
        WHERE DEPTNO = NEW.deptno;
        IF deptlabel != NEW.rowlabel
        THEN RAISE label_mismatch;
END;
```

VALUE CONSTRAINTS

Value constraints are defined integrity rules that may restrict the valid values for a data element. A value constraint defines a domain for a data attribute or set of attributes. Value constraints or domains range from simple datatypes (NUMBER,CHAR), to more complex expressions (SALARY > 0), to arbitrary business rules. The value constraints available in ANSI SQL Addendum I correspond closely to a SQL "WHERE" clause; SQL triggers are often used to implement more complex business rules over multiple tuples.

In the simplest case, all value constraints are defined at object creation and every value inserted meets all defined constraints. This simple case poses no security risk because the constraint only affects the subject doing the insert or update. In a more realistic environment, it is possible that additional constraints may be defined after data has been entered. A conflict arises when some existing data classified above the level of the user activating the constraint does not meet the new constraint.

One approach is to allow the constraint to be enabled if all visible data meets the constraint, and to only enforce the new constraint for all future data. The constraint might be considered fully enabled for all data visible to the user adding the constraint, and may or may not be considered enabled at higher access classes depending on the data values already present. This approach maintains security, but compromises data integrity since applications can no longer be guaranteed that all data meets all constraints.

Another approach is to notify the user that the constraint could not be enabled when some higher level data element is found which does not meet the constraint. This maintains integrity, but notifies the user of the existence of some higher level data. The degree of the security compromise is directly proportional to the flexibility of the constraint expression. A flexible constraint language such as that defined in SQL would allow the complete disclosure of the contents of all higher level data by successive attempts to enable constraints. The following constraint determines whether or not Allen makes more than $100,000.

```
ALTER TABLE EMP
        ADD (CHECK (SALARY > 100000
        OR ENAME != 'ALLEN'))
```

The exposure of this approach might be minimized by tightly restricting the ability to define new constraints (a sensible precaution anyway), and by the judicious use of auditing.

Finally, one might require that value constraints only be added by a MAC-privileged user with the ability to read all data values. The potential loss of flexibility is that an unprivileged user would be unable to define new constraints, even if the table actually contained no classified data, since the presence or absence of data are both signalling channels. Again, the security exposure can be reduced by restricting the operation, and by auditing possible attacks.

In a more complex model, value constraints may be defined at any access class, and any constraint applies only to data which is dominated by the constraint. Thus a constraint defined at the access class of the object definition might apply to all data, but a constraint defined at a higher access class would apply only to higher level data. This results in an obvious inconsistency and integrity violation – a user at the highest access class would observe that not all visible data would meet all visible constraints.

Triggers can be used to define more complex rules, such as the following trigger which attempts to enforce the rule that the total company payroll cannot exceed $200,000.

```
CREATE TRIGGER CASH_FLOW
        AFTER INSERT OR UPDATE OF SALARY ON EMP
        FOR EACH ROW
DECLARE
        total_salary EMP.SALARY%TYPE
        payroll_too_high EXCEPTION;
BEGIN
        SELECT SUM(SALARY) INTO total_salary
        FROM EMP;
        IF total_salary > 200000
        THEN RAISE payroll_too_high;
END;
```

If the rule is meant to enforce the rule across all access classes, then there are two possible options. The first option is to allow the trigger access to all data, including data above the user's access class. This approach is flawed, since the trigger might be used to leak information to the user.

The second option is to only allow the trigger access to data readable by the user. This eliminates the security problem by preventing full enforcement of the rule. In the example above, the trigger would only raise an exception where the total salaries visible to the user exceeded $200,000. An unclassified user would be able to add total salaries up to the full $200,000 regardless of any salaries at higher access classes. The trigger would raise an exception for the first user with a clearance high enough to see the rule violation.

This behaviour of MLS triggers illustrates the important difference between the use of triggers and constraints for enforcing integrity rules. The utility of declarative constraints, including primary and foreign key and value or domain constraints, is that the constraint declares an invarient property, detailing the meaning of a consistent state. If any declarative constraint is violated, the database is not in a consistent state. Applications can safely rely on declarative constraints. Triggers, however, are procedural and event-based. The existance of a trigger says nothing about the current or consistent state of a database, but merely specifies what happens during a transition from one state to the next.

A third option is to define rules to reference only a single access class. The earlier example might be split into two separate rules, one that states that unclassified salaries cannot exceed $150,000, and another stating that sensitive salaries cannot exceed $50,000. This satisfies both integrity and security, but requires additional effort in design and development.

TRANSACTION INTEGRITY

The transaction integrity property requires that all steps in a transaction be committed as an atomic unit, so that either all parts are committed or all parts are rolled back. A security conflict may arise if a single transaction is designed so that some steps must perform write operations at different access classes.

One example of such a transaction arises when a relationship between elements in an entity is classified. One way of classifying the relationship is to implement the entity with two tables, and the relationship with a third table. The example below shows the EMP table implementation suggested by [8] where employees and salaries are both unclassified, but the association of an employee with a salary is sensitive.

```
CREATE TABLE EMP(
        EMPNO NUMBER PRIMARY KEY,
        ENAME CHAR(20) NOT NULL)

CREATE TABLE SAL(
        S# NUMBER PRIMARY KEY,
        SALARY NUMBER)

CREATE TABLE EMPSAL (
        EMPNO NUMBER REFERENCES EMP (EMPNO),
        S# NUMBER REFERENCES SAL (S#))
```

Adding a new employee, an operation which might often be considered a single logical transaction, requires inserting unclassified records in the EMP and SAL tables, and a sensitive record in the EMPSAL table to make the sensitive association between the employee and his salary.

The simplest approach is to eliminate multilevel transactions from the application. This might be done by splitting a single logical transaction into individual transactions at each access class. The transaction above to add an employee might be processed as separate unclassified and sensitive transactions. The drawback is that splitting transactions may necessitate a redefinition of a consistant database state: between the two transactions a sensitive user will see an employee with no salary and a salary with no employee. This approach may be forced on a transaction if the database supports neither changing the access class of an existing subject nor any form of writeup. The remaining approaches depend on one or both these capabilities or on some knowledge of what data is read and written by the transaction.

If the highest access class data read by a transaction is dominated by the lowest data written, than the transaction may be carried out at the highest required read access class and all write steps may be carried out as write-up operations. Of course, SQL operations such as DELETE and UPDATE themselves do not qualify as pure write operations, since the data to be modified must first be read. Also, while write-up may not present a threat to operating system security, a database application may need to limit write-up for integrity reasons or for indirect security reasons. Applications often implicitly associate a higher access class with greater integrity; a write-up by a low user may violate that concept.

The obvious security flaw is that even a "pure" write-up done by a SQL INSERT may compromise security by inserting high values which test the enforcement of entity or referential integrity. The following insert by an unclassified user with the ability to write-up simply tests for the existence of employee number 1234 at sensitive.

```
INSERT INTO EMP (ROWLABEL,EMPNO)
       VALUES ('SENSITIVE',1234)
```

If the access classes written by a transaction can be totally ordered from low to high and the read steps are compatible with that ordering (i.e., no write step depends on a prior read step at a higher access class), then the transaction could securely move to the next higher access class as each step requires. Since the transaction's write access class is only permitted to change to a dominated access class, the write operations themselves disclose no information. But closer scrutiny shows that there is still information flow in whether or not the transaction has been successfully committed. While the bandwidth of the information channel is only a single bit, this is a useful channel since that one bit may easily be chosen to encode any information.

Finally, if the transaction is tightly controlled, it might be allowed to execute with privilege to circumvent mandatory access control. This, however, requires that either the transaction itself become part of the TCB, or that the user executing the transaction be properly authorized for such privileges. As the above discussion makes clear, even the normally innocuous write-up privilege poses a security threat in the context of integrity enforcement.

CONCLUSION

It is not possible to obtain both perfect data integrity and perfect multilevel security. Furthermore, the industry trend is towards supporting greater integrity enforcement capabilities. A variety of methods have been described here which show the extremes of maintaining either complete integrity or the prospect of complete security, as well as some intermediate tradeoff points which may allow greater latitude to particular applications in choosing an appropriate balance. Integrity is one of the principle benefits of a DBMS, but the use of the integrity features must be carefully scrutinized for non-obvious security flaws.

REFERENCES

[1]Burns, R.K., "Referential Secrecy." *IEEE Computer Security Symposium*, Oakland, CA, 133-142 (1990).

[2]"Database Language SQL with integrity enhancement" ANSI document X3H2-89-27.

[3]Melton, J. ed., Database Language SQL2 and SQL3, ANSI X3H2-090-062 (1990).

[4]*Department of Defense Trusted Computer System Evaluation Criteria*, Department of Defense, National Computer Security Center, DOD 5200.28-STD, December, 1985.

[5]Sandhu, R.S., Jajodia, S. and Lunt, T. "A New Polyinstantiation Integrity Constraint for Multilevel Relations." *IEEE Workshop on Computer Security Foundations*, Franconia, NH, 159-165 (1990).

[6]Lunt, T. and Hseigh, D. "Update Semantics for a Multilevel Relational Database." *IFIP WG 11.3 Workshop on Database Security*, Halifax, U.K. (1990).

[7]Jajodia, S., Sandhu, R., and Sibley, E. "Update Semantics for Multilevel Relations." *6th Computer Security Applications Conference*, Tucson, AZ, 103-112 (1990).

[8]Lunt, T. "Aggregation and Inference: Facts and Fallacies". *IEEE Symposium on Security and Privacy*, Oakland, CA, May 1989.

USING POLYINSTANTIATION IN THE REAL WORLD

Chip Paradise
The MITRE Corporation
7525 Colshire Drive
McLean, Virginia 22102

This paper describes an unusual implementation of polyinstantiation that addresses key functional and security requirements while avoiding several major disadvantages typically associated with a polyinstantiated database.

1 Background

The United States Transportation Command Military Airlift Command (USTRANSCOM/MAC) is a testbed for the Joint Multilevel Security (MLS) Technology Insertion Program (TIP). The overall goal of the MLS TIP is to develop and deploy technologies and components that will allow the Department of Defense (DOD) to meet operational and MLS requirements for their command and control systems. As a testbed for MLS technology, USTRANSCOM/MAC is playing a lead role within DOD in attempting to field an operationally usable, B1-certifiable MLS system, referred to as the Command and Control MLS (C^2MLS) Prototype.[1] A key characteristic of this system is its deliberate use of polyinstantiation.[2]

2 Environment

Two single-level, system high systems at USTRANSCOM/MAC manage flight schedules, aircraft location information, readiness data, cargo data, and supplies and fuel requirements. One system operates at the Sensitive Unclassified level while the other operates at the Secret level. A cleared user must retrieve data from both systems to obtain a complete status of USTRANSCOM/MAC assets.

The C^2MLS Prototype is being developed to provide USTRANSCOM/MAC command center personnel with a single unified view of data and to reduce system duplicity. This prototype is targeted to manage Sensitive Unclassified and Secret information, and to permit online access for both cleared and uncleared users. A B1-targeted database management system (DBMS) and operating system, the SYBASE Secure SQL Server and DEC's Security Enhanced Virtual Memory System (SE/VMS), respectively, are being integrated into the system architecture.[3] A trusted networking component is scheduled to be selected and integrated into the prototype in 1992.

---

1 Digital Equipment Corporation (DEC) is under contract to USTRANSCOM/MAC to serve as the integration and development contractor for the Prototype (Contract Number F11623-89-D0007). MITRE is under contract to the Defense Information Systems Agency (DISA) to provide security engineering support to USTRANSCOM/MAC (Contract Number DAAB07-91-C-N751).

2 Polyinstantiation is the simultaneous existence in a database of different versions of data with the same primary key, but at different security levels.

3 At the time this paper was written, the National Computer Security Center (NCSC) had not completed a formal evaluation of either product.

3 Key Requirements

Several key USTRANSCOM/MAC data management and data display requirements had a significant affect on the design and management of polyinstantiation in the prototype. These requirements were:

- Support cover stories in order to provide uncleared users with sufficient data to perform their jobs, while protecting the mission's Secret data and preventing inferences about that data.
- Maintain data element labels to facilitate users making element-level disclosure decisions from their terminals.
- Support screen formats that are similar to those supported by the system high systems to reduce the training required to transition to MLS operations [DONC90]. Such formats require that polyinstantiated tuples be collapsed into a single composite representation prior to being displayed.
- Define and enforce classification constraints to control the sensitivity of data values assigned to data elements, and to control the labeling of data elements comprising a data aggregate within a tuple.

These requirements have been satisfied by the C^2MLS Prototype. Two requirements, cover stories and data element labels, are directly supported through polyinstantiation. Another requirement, single composite representations of polyinstantiated tuples, is supported by collapsing polyinstantiated tuples into a single logical structure. The remaining requirement, classification constraints, strictly controls the data composition of polyinstantiated tuples. All solutions are implemented with trusted application software. Implementation details appear in the following paragraphs.

4 Implementation Details

4.1 Cover Stories

In the context of the USTRANSCOM/MAC C^2MLS effort, a cover story describes a minimal set of information that is sufficient for uncleared users to perform their job. An example of a cover story is Sensitive Unclassified information indicating a plane is on a mission. The fact that a plane is on a mission must be known to USTRANSCOM/MAC flight planners and schedulers operating in a Sensitive Unclassified session, so they do not attempt to schedule the plane for another mission. While the data indicating the plane is on a mission are factual, mission-specific data elements may contain values that are fictitious. The actual or "true" mission-specific details are stored in a Secret tuple. The need for multiple tuples to completely describe all data about a mission is readily supported through polyinstantiation.

In the USTRANSCOM/MAC C^2MLS prototype, polyinstantiation is supported through trusted application software and not through the DBMS. This trusted application software enforces several restrictions on polyinstantiation. The Sensitive Unclassified tuple is conceptually the foundation upon which a Secret tuple is built, such that a Secret tuple is created only through the polyinstantiation of a Sensitive Unclassified tuple. Additionally, the Secret tuple contains only the primary key value and the data element values that were updated by the Secret user. The remaining columns contain null values. This is done to minimize data replication within the database. (This is the case for most but not all relations in the database. Exceptions include relations that hold itinerary data and remarks information. In these few instances,classified tuples are allowed to exist without underlying Sensitive Unclassified tuples.)

4.2 Data Element Labels [4]

An objective of the the prototype is to provide trusted output labeling, such that for retrieval, data element labels are dominated by the user's session level, and for update, the data element label is equal to the user's session level [4]. In order to realize data element labels at the user and program interfaces, the DBMS should ideally maintain them. However, label granularity for the SYBASE Secure SQL Server is at the tuple level. This dichotomy between product capability and functional requirement has been addressed by minimizing replicated data in a Secret tuple such that, except for the primary key, only Secret data reside in a Secret tuple. Thus, trusted application software can derive data element labels from tuple labels.

4.3 Classification Constraints

In the C^2MLS prototype, classification constraints specify the range of sensitivity levels for columns and groups of columns in a table. Classification constraints are defined in metadata extensions and enforced by trusted application software. Metadata extensions define each data element as one of three types: fixed, swing, or critical swing. Fixed data elements may contain only Sensitive Unclassified values. In Figure 1, the data elements "Sched Dest" and "Sched DTG" are fixed. Swing data elements may contain either Sensitive Unclassified or Secret values. A critical swing data element may contain either Sensitive Unclassified or Secret values; when it contains Secret data, it causes a predefined set of other swing data elements, referred to as a swing group, to be reclassified as Secret. In Figure 1, Polyinstantiated Tuple, the data element "Actual Dest" is a critical swing data element and is associated with the swing data element, "Actual DTG."

A Sensitive Unclassified tuple contains fixed data values and optionally, swing and critical swing values. If a polyinstantiated Secret tuple exists, the swing and critical swing values in the Sensitive Unclassified tuple represent a cover story. Blank values are permitted for swing and critical swing values in Sensitive Unclassified tuples. In Figure 1, the values assigned to "Actual Dest" and "Actual DTG" in the Sensitive Unclassified tuple (designated by the tuple label (TL) "U") are cover stories.

A Secret tuple will exist if there are Secret values for critical swing and swing data elements. Sensitive Unclassified fixed values are not replicated in the Secret tuple; instead, null values are stored. In Figure 1, when "Actual Dest" is changed to "Baghdad," a Secret tuple (designated by the tuple label "S") is created. The primary key is copied from the "U" tuple, null values are assigned to the fixed data elements, "Baghdad" becomes the value for the critical swing data element "Actual Dest," and the value for "Actual DTG" is replicated and stored as a Secret value because of its association with "Actual Dest."

4.4 Single Composite Representation of Data

When a Secret user retrieves data, trusted application software combines polyinstantiated tuples and builds a composite representation, called labeled data objects (LDOs), for display to the user. Tuples are retrieved in the order of their sensitivity from Sensitive Unclassified to Secret. As shown in Figure 2, Labeled Data Objects, trusted application software overlays the Sensitive Unclassified data in the composite representation with the Secret data, and data element labels are generated from tuple labels (data element labels are shown under the "FL" columns). Null values are ignored and do not overlay data values. Users operating at the Sensitive Unclassified level see only data from the Sensitive Unclassified tuple, and users operating at a Secret level see a single composite representation of data as represented in the LDOs. This approach is reasonable for the

[4] In the current implementation of the C^2MLS Prototype, data element labels that are displayed to users, must be considered advisory since several trusted components, including trusted networking and display management, are not integrated into the Prototype.

		FIXED	FIXED	CRITICAL SWING	SWING
TL	MISSION ID	SCHED DEST	SCHED DTG	ACTUAL DEST	ACTUAL DTG
U	123A	TEL AVIV	0830	TEL AVIV	0900

Existing Sensitive Unclassified Tuple

S-USER:
Change ACTUAL DEST To "BAGHDAD"

		FIXED	FIXED	CRITICAL SWING	SWING
TL	MISSION ID	SCHED DEST	SCHED DTG	ACTUAL DEST	ACTUAL DTG
U	123A	TEL AVIV	0830	TEL AVIV	0900
S	123A	NULL	NULL	BAGHDAD	0900

After Trusted Application Software Inserts Secret Tuple

Figure 1. Polyinstantiated Tuple

USTRANSCOM/MAC environment because Secret data is considered more accurate and to have greater validity and importance than Sensitive Unclassified data.

5 Design Advantages

The prototype design satisfies the requirements outlined in Section 3, and offers several advantages over other designs incorporating polyinstantiation. These advantages include:

- Improved database integrity and accurate data element labels
- Reduced user confusion
- Reduced disk space usage

One of the problems with replicating Sensitive Unclassified data in Secret tuples is the update transaction, that replicates the data, may fail. If transaction controls are inadequate to react and restart the update, the database becomes inconsistent. A second problem with replicating Sensitive Unclassified data in Secret tuples is the difficulty of preventing cleared users from updating the Sensitive Unclassified data in the Secret tuple. Such updating results in the data migrating upwards in classification. The C^2MLS prototype design addresses these database integrity and labeling issues by replicating only the Sensitive Unclassified primary key across tuples,

		FIXED	FIXED	CRITICAL SWING	SWING
TL	MISSION ID	SCHED DEST	SCHED DTG	ACTUAL DEST	ACTUAL DTG
U	123A	TEL AVIV	0830	TEL AVIV	0900
S	123A	NULL	NULL	BAGHDAD	0900

Two Tuples As Stored In The DB

<u>Trusted Application Software</u>:
Builds Single Composite Record, or LDOs, From Secret and
Sensitive Unclassified Tuples

		FIXED		FIXED		CRITICAL SWING		SWING	
FL	MISSION ID	FL	SCHED DEST	FL	SCHED DTG	FL	ACTUAL DEST	FL	ACTUAL DTG
U	123A	U	TEL AVIV	U	0830	S	BAGHDAD	S	0900

Labeled Data Objects (Display & Internal Use Only - Not Stored)

Figure 2. Labeled Data Objects

and enforcing data element classification constraints to prevent the data values from migrating upwards in classification.

Since trusted application software builds a single composite representation of Sensitive Unclassified and Secret data, cleared users do not need to view multiple tuples to ascertain the complete status of an entity. This single composite representation of data permits screen formats from the current system high system to be retained. By retaining these screens, users will continue to interact with a user interface that is familiar to them, even though the actual underlying system has changed significantly.

Lastly, the amount of disk space storage for the database is reduced. In the MLS prototype design, a null value is stored for each fixed data element in a Secret tuple. By storing a null value instead of variable length text values, reasonable disk space savings can be expected when compared to other polyinstantiated designs with significant data replication.

6 Design Disadvantages

While the MLS prototype satisfies the requirements outlined in Section 3, there are several disadvantages associated with this design. These disadvantages are:

- Size and complexity of the trusted computing base (TCB)
- Hierarchical sensitivity levels and data validity rules
- Overhead to build composite representations of data

In the prototype design, trusted application software had to be developed to provide MLS services that were not supported by the trusted COTS software. Examples of these services include composite displays, label management, polyinstantiation with minimal data replication, and the specification and enforcement of classification constraints. Since many of these services enforce various portions of the C^2MLS security policy [DCA88], the trusted application software significantly extends the system TCB. The trusted application software will need to be subjected to rigorous certification analysis and testing to validate its correct execution.

A second disadvantage associated with this design is that it is viable only in those systems with hierarchical sensitivity levels and a hierarchical correspondence to the validity of data. For example, if a Secret tuple exists, the Secret values are assumed to have greater validity than the Sensitive Unclassified values, and the values appear in place of Sensitive Unclassified values in a composite representation.

Another disadvantage specific to this design is the processing overhead incurred in building composite representations of data from polyinstantiated tuples. Composite representations of data are built to minimize the number of tuples a cleared user must inspect to ascertain all information about an entity, and to permit screen formats from the current operational systems to be retained.

7 Deletion and Downgrading Anomalies

While this implementation of polyinstantiation has overcome several shortcomings, several processing anomalies have been introduced. Two such anomalies are associated with tuple deletion and downgrading.

The tuple deletion anomaly is associated with Sensitive Unclassified tuples. For example, assume that a Sensitive Unclassified tuple and a polyinstantiated Secret tuple exist in the database. If an uncleared user is allowed to delete the Sensitive Unclassified record, the null values in the Secret tuple can no longer be resolved. If an uncleared user is prevented from deleting the Sensitive Unclassified tuple because of the existence of the Secret tuple, the uncleared user can infer that the Secret tuple exists. The prototype's interim solution to deleting Sensitive Unclassified tuples allows both cleared and uncleared users to request the deletion of Sensitive Unclassified tuples. If a Secret tuple exists, the user is informed that the delete operation was denied. As the prototype matures, the developers will attempt to eliminate this inference channel for uncleared users.

The anomaly associated with the prototype's downgrading approach is the need to create a new tuple instead of re-labeling an existing one. Since both tuples contain information that must be present in a new downgraded tuple, a Sensitive Unclassified tuple and its polyinstantiated Secret tuple must be collapsed into a single composite representation. This composite representation replaces the current Sensitive Unclassified tuple.

An issue associated with the downgrading approach is whether the COTS DBMS can support required downgrading functionality. USTRANSCOM/MAC's approach for assigning a sensitivity label to mission data is to reduce the sensitivity of mission data as the mission progresses. As a mission is planned, data supporting the mission are classified at a very restrictive level. As a mission gets under way and then is completed, the information's sensitivity is reduced until it becomes unclassified. To facilitate a downgrading approach that mirrors USTRANSCOM/MAC's mission planning and execution steps, a trusted DBMS must support a set-based multi-table downgrade utility. Since the initial release of the SYBASE Secure DBMS downgrades an entire table, a set-based approach had to be implemented with trusted application software. This trusted application software coordinates the execution of both Sensitive Unclassified and Secret processes to retrieve the old tuples, build the new composite representation, update the Sensitive Unclassified tuple, and delete the old Secret tuple.

8 Summary

This paper has described a prototype implementation where polyinstantiation has been deliberately used to meet several operational requirements. Its design is such that commonly considered handicaps have been somewhat overcome and important advantages can be identified. Conversely, this design has introduced several issues that require analysis and resolution. As prototyping continues, reasonable solutions will hopefully be identified and implemented.

Acknowledgements

The author appreciates the technical exchange of ideas that took place early in the project with Victoria Ashby of The MITRE Corporation, the advice and review provided by Mindy Rudell and LouAnna Notargiacomo of The MITRE Corporation, and the encouragement offered by Dr. Sushil Jajodia of George Mason University.

List of References

[DCA88] Defense Communications Agency (DCA), September 1988, *Security Policy for the Military Airlift Command's Command and Control System (Draft)*, DCA Technical Report (TR)-xx, Washington, DC.

[DONC90] Samuel Doncaster, Michael Endsley, Greg Factor,"Rehosting Existing C^2 Systems Into An MLS Environment," *Proceedings from the Sixth Annual Computer Security Applications Conference*, Tucson, Arizona, December, 1990.

[WILS90] William Wilson, Joel Sachs, David Wichers, Peter Boucher, December 1990, "MLS and Trust Issues at the User Interface in MLS AISs," *Proceedings from the Sixth Annual Computer Security Applications Conference*, Tucson, Arizona.

ENFORCING PRIMARY KEY REQUIREMENTS IN MULTILEVEL RELATIONS

Sushil Jajodia and Ravi S. Sandhu

Center for Secure Information Systems
and
Department of Information and Software Systems Engineering
George Mason University, Fairfax, VA 22030-4444

1 INTRODUCTION

The notion of a primary key is considered a fundamental concept in the classical (single-level) relational model. For example, it forms the basis for several normal forms and is used when the database schema is designed. The primary key is used to maintain integrity of relations. It is also used for storage and retrieval purposes.

Unfortunately, the concept of a primary key does not extend to multilevel relations in a straightforward way because of two factors: (a) the $\star$-property must be preserved which prevents any write downs, and (b) signaling channels must be avoided. These security considerations have lead to the notion of *polyinstantiation* in multilevel relations [2].

Polyinstantiation comes in several different flavors [2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]. There are significant differences between these approaches and debate continues about the correct definition of polyinstantiation and its operational semantics. However, in each case polyinstantiation fails to preserve the basic requirement of a primary key that there be one and only one tuple per primary key value in a relation. Polyinstantiation forces a relation to contain multiple tuples with the same primary key, distinguishable by their classifications or by non-primary key attribute values.

Since polyinstantiation significantly complicates the semantics of multilevel relations (particularly for high users), recently some solutions have appeared which attempt to do away with polyinstantiation completely [1, 13, 14]. In this paper, we take another step along this direction, and examine ways to preserve primary key requirements in multilevel relations. Of course, any solution we give will have to be secure and free of denial-of-service problem.

The organization of the remainder of this paper is as follows. In section 2 we briefly review the notion of primary key in classical (single-level) relations. In section 3 we show how polyinstantiation arises in multilevel relations. In section 4 we explore alternatives to polyinstantiation that help us enforce primary key requirements in multilevel relations. Finally, the conclusion is given in section 5.

2 PRIMARY KEY IN SINGLE-LEVEL RELATIONS

The standard relational model is concerned with data without security classifications. Data are stored in relations that have well-defined mathematical properties. Each relation has two parts as follows.

1. A state-invariant *relation scheme* $R(A_1, A_2, \ldots, A_n)$, where each A_i is an *attribute* over some *domain* D_i which is a set of values.

2. A state-dependent *relation* r over R, which is a set of distinct *tuples* of the form $(a_1, a_2, \ldots, a_n)$ where each *element* a_i is a value in domain D_i.

Starship		Objective		Destination		TC
Enterprise	U	Exploration	U	Talos	U	U
Voyager	S	Spying	S	Mars	S	S

Figure 1: SOD_S

Starship		Objective		Destination		TC
Enterprise	U	Exploration	U	Talos	U	U

Figure 2: SOD_U

Not all possible relations are meaningful in an application; only those that satisfy certain integrity constraints are considered valid.

The notion of primary key is central to the relation model. Before we define it, however, we need to give a definition for a candidate key:

We say $X \subseteq \{A_1, A_2, \ldots, A_n\}$ is a *candidate key* of R if any relation r for R at all times satisfies the following two properties:

Uniqueness Property. The relation r does not contain two distinct tuples with the same values for X.

Minimality Property. No proper subset Y of X satisfies this uniqueness property.

A *primary key* of a relation scheme R is a candidate key of R. It is possible that a relation scheme has more than one candidate key, in which case one of the candidate keys must be chosen and designated as the primary key.

3 POLYINSTANTIATION

While the notion of a primary key is simple and well understood for classical (single-level) relations, it does not have a straightforward extension to multilevel relations. To illustrate, consider the relation scheme SOD(Starship, Objective, Destination) where Starship is the primary key and the security classifications are assigned at the granularity of individual data elements. Suppose the Secret and Unclassified views of SOD are as shown in figures 1 and 2, respectively.

Suppose that an U-user* who sees the instance in figure 2 wishes to insert a second tuple (Voyager, Exploration, Talos) to SOD_U. If we were to enforce the primary key requirement, this insertion by the U-user will be rejected (since it conflicts with an existing tuple in SOD_S). However, since this rejection will create a signaling channel, both tuples (Voyager, Spying, Mars) and (Voyager, Exploration, Talos) are allowed to co-exist in SOD_S, as in figure 3, in violation of the uniqueness requirement. This is one type of polyinstantiation, called *entity polyinstantiation*: A relation contains two or more tuples with the same primary key values, but having different access class values for the primary key.

There is another form of polyinstantiation, called *element polyinstantiation*. With element polyinstantiation, a relation contains two or more tuples with identical primary key and the associated access class values, but having different non-primary key values, as shown in the relation in figure 4. The objectives and destinations of the starship Enterprise are different for U- and S-users.

*Strictly speaking we should be saying subject rather than user.

Starship		Objective		Destination		TC
Enterprise	U	Exploration	U	Talos	U	U
Voyager	U	Exploration	U	Talos	U	U
Voyager	S	Spying	S	Mars	S	S

Figure 3: SOD$_S$

Starship		Objective		Destination		TC
Enterprise	U	Exploration	U	Talos	U	U
Enterprise	U	Spying	S	Rigel	S	S

Figure 4: SOD$_S$

Both entity and element polyinstantiation can occur in basically two different ways:

1. A high user attempts to update data which conflicts with the existing low data. Since overwriting the low data in place will result in a downward signaling channel, the tuple is polyinstantiated
2. An opposite situation occurs where a low user attempts to insert data which conflicts with existing high data. Since rejecting the update is not a viable option because it establishes a downward signaling channel, the updated tuple is polyinstantiated to reflect the low update.

4 PRIMARY KEY IN MULTILEVEL RELATIONS

In this section, we explore how we can enforce primary key requirements in multilevel relations without creating a downward signaling channel in the process.

4.1 A Simple Solution

There is a completely obvious way to preserve primary key requirements in multilevel relations.

1. Whenever a high user makes an update which violates the uniqueness requirement, we simply refuse that update.
2. Whenever a low user makes a change with conflicts with the uniqueness requirement, the conflicting high data is *overwritten in place* by the low data.

Returning to the example of the previous section, when the U-user inserts the second tuple (Voyager, Exploration, Talos) to SOD$_U$ shown in figure 2, the conflicting second tuple in SOD$_S$ in figure 1 is substituted by the newly inserted tuple. As a result both U- and S- users see the instance shown in figure 5. On the other hand, suppose a S-user wants to insert the tuple (Voyager, Spying, Mars) to SOD$_S$ in figure 5. This update is simply refused. Thus, in both cases primary key values in SOD uniquely identify the tuples in the multilevel relations.

It is not difficult to see that this simple solution preserves the uniqueness requirement in multilevel relations. This solution is secure in the sense of secrecy and information flow. It is our view that

Starship		Objective		Destination		TC
Enterprise	U	Exploration	U	Talos	U	U
Voyager	U	Exploration	U	Talos	U	U

Figure 5: $SOD_S = SOD_U$

while this solution may be acceptable in some specific situations, it is clearly unacceptable as a general solution; it can lead to serious denial-of-service and integrity problems. Therefore, we now look for other alternatives which do not suffer from these problems.

4.2 Dealing with Entity Polyinstantiation

4.2.1 Single Access Class for the Primary key

A multilevel relation is created by using a data definition statement, similar to the following statement:[†]

```
CREATE TABLE SOD ( Starship     CHAR(15) NOT NULL [U:S],
                   Objective    CHAR(15) {U, TS},
                   Destination  CHAR(20) [U:TS],
                   Primary Key (Starship ) );
```

Here the domain of the access class of the primary key Starship has been specified as a range with a lower bound of U and an upper bound of S. As we saw in the previous section, this leads to entity polyinstantiation. Thus, one simple way of eliminating entity polyinstantiation is to have the domain of the access class of the primary key consist of a single element.

Thus if we create the SOD relation as follows, SOD will not have any entity polyinstantiation.

```
CREATE TABLE SOD ( Starship     CHAR(15) NOT NULL {U}
                   Objective    CHAR(15) {U, TS},
                   Destination  CHAR(20) [U:TS],
                   Primary Key (Starship ) );
```

It is possible that in some situation names of some starships must remain Top Secret, in such a case we can use the following solution.

4.2.2 Partitioning the Domain of the Primary Key

Another way to eliminate entity polyinstantiation is to partition the domain of the primary key among the various access classes possible for the primary key. For our example, we can introduce a new attribute, called Starship#. Whenever a new tuple is inserted, we enforce the requirement that all the Starships numbered between 1 and 1,000 will be unclassified, those numbered between 1,001 and 2,000 will be confidential, and so on.

In SQL-like language, the SOD schema should be created as follows:

[†]The notation [L:H] specifies a range of security classes with lower bound L and upper bound H. The notation {X,Y,Z} enumerates the allowed values for the security class as one of X, Y or Z.

```
CREATE TABLE SOD
  ( Starship#     SMALL INTEGER NOT NULL [U:TS]
    Starship      CHAR(15) NOT NULL [U:TS]
    Objective     CHAR(15) {U, TS},
    Destination   CHAR(20) [U:TS],
    Primary Key (Starship# ),
    CHECK (User Access class = 'U'  AND Starship# BETWEEN 1    AND 1000),
    CHECK (User Access class = 'C'  AND Starship# BETWEEN 1001 AND 2000),
    CHECK (User Access class = 'S'  AND Starship# BETWEEN 2001 AND 3000),
    CHECK (User Access class = 'TS' AND Starship# BETWEEN 3001 AND 4000) );
```

4.3 Dealing with Element Polyinstantiation

It is possible to eliminate element polyinstantiation securely without sacrificing either integrity or availability. We show how this is done for the SOD example. For complete details, we refer the reader to [13]. This solution by Sandhu and Jajodia meets the following requirements.

1. There are no downward signaling channels.
2. The simple security and the $\star$-properties is enforced for all subjects, i.e., no trusted code can be used.
3. There are no temporary inconsistencies.
4. There is no denial of data entry service to high users.

Consider once again the following relation SOD where Starship is the primary key. We assume that the Starship attribute is always unclassified, so there is no entity polyinstantiation.

Starship		Objective		Destination		TC
Enterprise	U	Exploration	U	Talos	U	U

Consider the following scenario. Suppose a S-user attempts to modify the destination of the Enterprise to be Rigel. We cannot polyinstantiate even temporarily, so we must reject this update. There is no denial-of-service to the S-user since the S-user can obtain service as follows.

Step 1. The S-user first logs in as a U-subject and marks the destination of the Enterprise as restricted giving us the following relation.

Starship		Objective		Destination		TC
Enterprise	U	Exploration	U	restricted	U	U

The meaning of restricted is that this field can no longer be updated by a U-user. U-users can therefore infer that the true value of Enterprise's destination is classified at some level not dominated by U.

Step 2. The S-user then logs in as a S-subject and enters the destination of the Enterprise as Rigel giving us the following relations at the U and S levels, respectively.

Starship		Objective		Destination		TC
Enterprise	U	Exploration	U	restricted	U	U

Starship		Objective		Destination		TC
Enterprise	U	Exploration	U	Rigel	S	S

One can argue that step 2 introduces a signaling channel. Fortunately, this is not a particularly harmful channel. Here a trusted S-user is in the loop who presumably will ensure that the channel is not exercised wantonly, but rather that this inference is permitted only when the real world situation is actually so. Such a channel with trusted humans in the loop can be exercised only by Trojan Horses that are capable of manipulating the real world. This entails the manipulation of real trusted people making real decisions and not merely the manipulation of bits in a database.

We refer the reader to [13] for additional examples and complete details.

5 CONCLUSION

In this paper we have shown that there are ways to enforce primary key requirements in multilevel relations. The methods we have listed eliminate problems that arise from polyinstantiation completely. These methods may be eminently suitable in many applications.

Yet we wish to remind the reader that there are situations where polyinstantiation is desirable. There is a real need for cover stories in the multilevel world, and polyinstantiation provides a simple way of satisfying this need. Moreover, we envision applications (particularly in an intelligence environment) where information is coming from different sources, bearing different classification. The information may sometimes be contradictory; however, it must be stored in the database. An analyst can make sense out of the confusing mess. It is desirable to have polyinstantiation for such situations.

References

[1] Rae K. Burns, "Referential Secrecy." *Proc. IEEE Symposium on Security and Privacy*, Oakland, California, May 1990, pages 133-142.

[2] Dorothy E. Denning, Teresa F. Lunt, Roger R. Schell, Mark Heckman, and William R. Shockley, "A multilevel relational data model." *Proc. IEEE Symposium on Security and Privacy*, April 1987, pages 220-234.

[3] J. T. Haigh, R. C. O'Brien, and D. J. Thomsen, "The LDV Secure Relational DBMS Model." *Database Security IV: Status and Prospects*, S. Jajodia and C. E. Landwehr (editors), North-Holland, 1991, pages 265-279.

[4] Sushil Jajodia and Ravi S. Sandhu, "Polyinstantiation integrity in multilevel relations." *Proc. IEEE Symposium on Security and Privacy*, Oakland, California, May 1990, pages 104-115.

[5] Sushil Jajodia and Ravi S. Sandhu, "A formal framework for single level decomposition of multilevel relations." *Proc. IEEE Workshop on Computer Security Foundations*, Franconia, New Hampshire, June 1990, pages 152-158.

[6] Sushil Jajodia and Ravi S. Sandhu, "Polyinstantiation integrity in multilevel relations revisited." *Database Security IV: Status and Prospects*, S. Jajodia and C. E. Landwehr (editors), North-Holland, 1991, pages 297-307.

[7] Sushil Jajodia, Ravi S. Sandhu, and Edgar Sibley, "Update semantics of multilevel relations." *Proc. 6th Annual Computer Security Applications Conf.*, December 1990, pages 103-112.

[8] Sushil Jajodia and Ravi S. Sandhu, "A novel decomposition of multilevel relations into single-level relations." *Proc. IEEE Symposium on Security and Privacy*, Oakland, California, May 1991, pages 300-313.

[9] Sushil Jajodia and Ravi S. Sandhu, "Toward a multilevel secure relational data model," *Proc. ACM SIGMOD Int'l. Conf. on Management of Data,* Denver, Colorado, May 29-31, 1991, pp. 50-59.

[10] Teresa F. Lunt, Dorothy E. Denning, Roger R. Schell, Mark Heckman, and William R. Shockley, "The SeaView security model." *IEEE Transactions on Software Engineering*, Vol. 16, No. 6, June 1990, pages 593-607.

[11] Teresa F. Lunt and Donovan Hsieh, "Update semantics for a multilevel relational database." *Database Security IV: Status and Prospects*, S. Jajodia and C. E. Landwehr, (editors), North-Holland, 1991, pages 281-296.

[12] Ravi S. Sandhu, Sushil Jajodia, and Teresa Lunt, "A new polyinstantiation integrity constraint for multilevel relations." *Proc. IEEE Workshop on Computer Security Foundations*, Franconia, New Hampshire, June 1990, pages 159-165.

[13] Ravi S. Sandhu and Sushil Jajodia, "Honest databases that can keep secrets," *Proc. 14th NIST-NCSC National Computer Security Conference*, Washington, D.C., October 1991, To appear.

[14] S. R. Wiseman, "On the Problem of Security in Data Bases." In *Database Security III: Status and Prospects*, (Spooner, D.L. and Landwehr, C.E., editors), North-Holland, 1990 pages 143-150.

Inference Through Polyinstantiation

Leonard J. Binns

Office of INFOSEC Computer Science

•

United States Department of Defense
Fort George G. Meade, Maryland

Abstract:

Polyinstantiation is allowed in most multilevel database systems to prevent a covert channel from opening up in an otherwise secure environment. It is also used in some systems to provide cover stories or "shades" of information. The latter uses of polyinstantiation, however, have the potential to open up inference channels allowing an unauthorized user to obtain classified information.

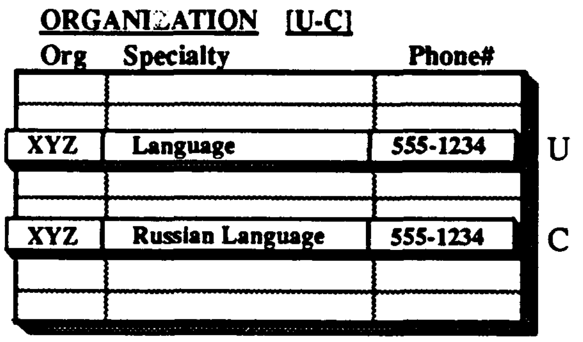
These findings were noted during research on inference control. The analysis we used to detect inference paths indicated potential inferences where none appeared to exist. This was attributed to the inability of the analysis to understand the concept of polyinstantiation, and was almost dismissed. Further investigation has now shown how polyinstantiation can, in some situations, provide an inference path where none previously existed.

This is the first time polyinstantiation has been linked with the potential downward flow of information in a multilevel secure system. Previously, polyinstantiation was only seen as a way to *prevent* the downward flow of information.

Introduction

Polyinstantiation has two primary purposes: prevent covert channels and provide cover stories[1]. The intentional introduction of polyinstantiation into a database falls under the second category. Consider the following scenario:

There exists an organization which interacts with some collection of foreign countries. Most of the related information is unclassified while some is sensitive. One branch of the organization (XYZ) has an unclassified specialty of "Language." The classified user, however, is shown more specifically that it's specialty is "Russian Language." This is illustrated below in figure 1.



ORGANIZATION [U-C]

Org	Specialty	Phone#	
XYZ	Language	555-1234	U
XYZ	Russian Language	555-1234	C

Figure 1

The user bases the classification of these tuples on the classification guideline listed in appendix A.

Does this schema and it's contents violate the security policy? Perhaps. If this is the only table in the database and if this information cannot be exported or joined with information existing elsewhere, then the answer is no. As more relations are created and information is allowed to flow in and out of our database system, the answer becomes more difficult to answer.

1. or "shades" of information, i.e. Smith is a Linguist at the unclassified level and a Russian Linguist at the confidential level.

This paper shows how the intentional introduction of polyinstantiation can itself lead to the disclosure of information it is attempting to protect; it is an illustration of "cover stories that tell the *truth.*"

The goal here is to address a specific type of inference; inference through polyinstantiation. This paper will identify:

- How the intentional introduction of polyinstantiation can lead to inference
- How it was discovered, and
- What is required to assure no inference is introduced by the polyinstantiation of a record.

Inference Analysis

Figures 2 and 3 set the stage for the inference analysis mechanism.

Figure 2 represents the existing database by the relations that are defined. This model allows for multilevel relations at the tuple level, where the container defines what range of classification is allowed within that relation. In this database there are three relations named Organization [U-C], Phonebook [U], and Education [U].

Figure 3 represents the database attributes and how they are associated. All arcs represent a direct association between two attributes. The arcs are labeled by the relation connecting each pair of attributes[2]. A path between two attributes (a and b) indicates that given *a*, there exists some set of joins which could potentially produce *b*. With weighted arcs,

2. In a more detailed representation, the arcs are both directed and weighted (0 to 1); thereby indicating more accurately the probability that one attribute will uniquely identify another. Example: org --(1)--> phone means org positively identifies phone, where phone --(.9)--> org indicates there is a 90% probability that phone will correctly imply org.

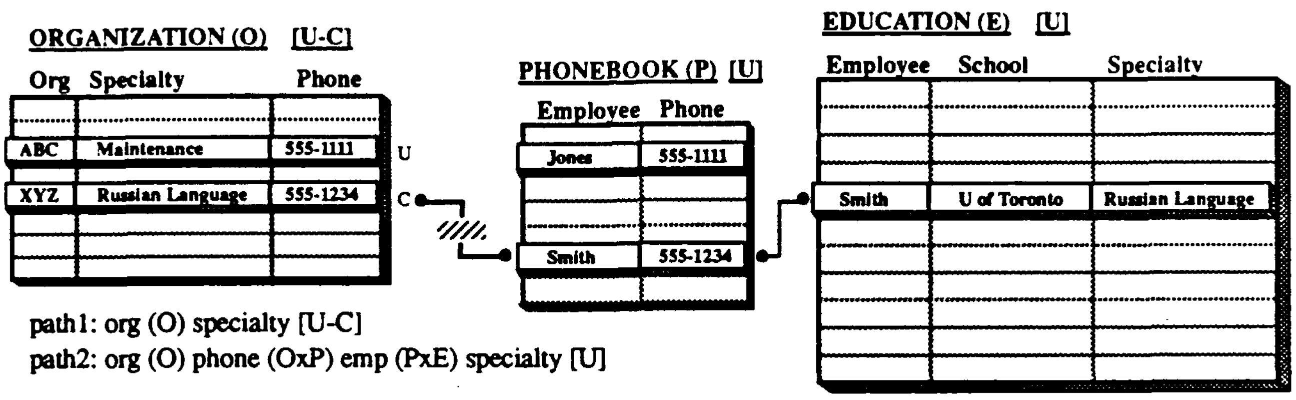

ORGANIZATION (O) [U-C]
Org Specialty Phone
ABC Maintenance 555-1111 U
XYZ Russian Language 555-1234 C
PHONEBOOK (P) [U]
Employee Phone
Jones 555-1111
Smith 555-1234
EDUCATION (E) [U]
Employee School Specialty
Smith U of Toronto Russian Language
path1: org (O) specialty [U-C]
path2: org (O) phone (OxP) emp (PxE) specialty [U]

Figure 2

this potential is described as a probability between 0 and 1. The system security officer determines what threshold (if any) the security policy will tolerate.

Consider the relationship between *org* and *specialty* in figures 2 and 3. Is there an inference problem between these two attributes in the present schema? By definition[1], an inference problem exists if there are multiple paths between two attributes such that the paths have potentially different classification levels. Looking at figure 2 we see that there are two paths from org to specialty. There is no inference problem because classification of a tuple in O prevents access to O.phone for the unclassified user, hence there is no way to join it with P.phone and the path using O, P and E is broken. In this model it appears that there cannot exist two paths from org to specialty having different classifications. Either there will exist one classified path, or two unclassified paths.

During analysis, however, a mechanism we had begun to prototype indicated an inference problem. The inference mechanism used the representation shown in figure 3. As shown in the graph there *are* two paths between org and specialty, and they have potentially different classifications. This result was almost dismissed as the mechanism's inability to recognize that the direct paths between org-specialty and org-phone are really represented by the same path (O). If the path between org and specialty through O is classified, then so is the path between org and phone. *Unless* we allow polyinstantiation.

How Polyinstantiation can Lead to Inference

Going with the idea that polyinstantiation could give us two differently classified paths between org-specialty and org-phone, the task was to determine how this could create a security hole. By our earlier definition, the existence of these two paths did create a potential inference path because now there was a

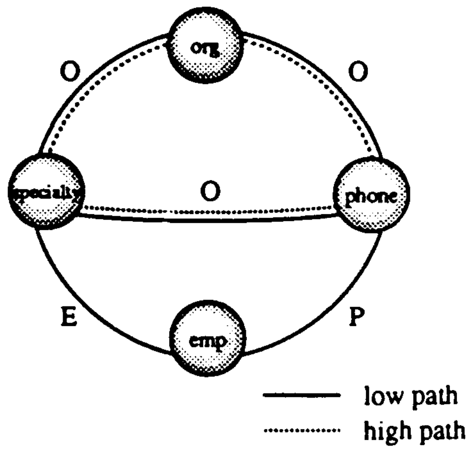


Figure 3

1. Appendix B formally definitions one class of inference: inference through secondary path analysis.

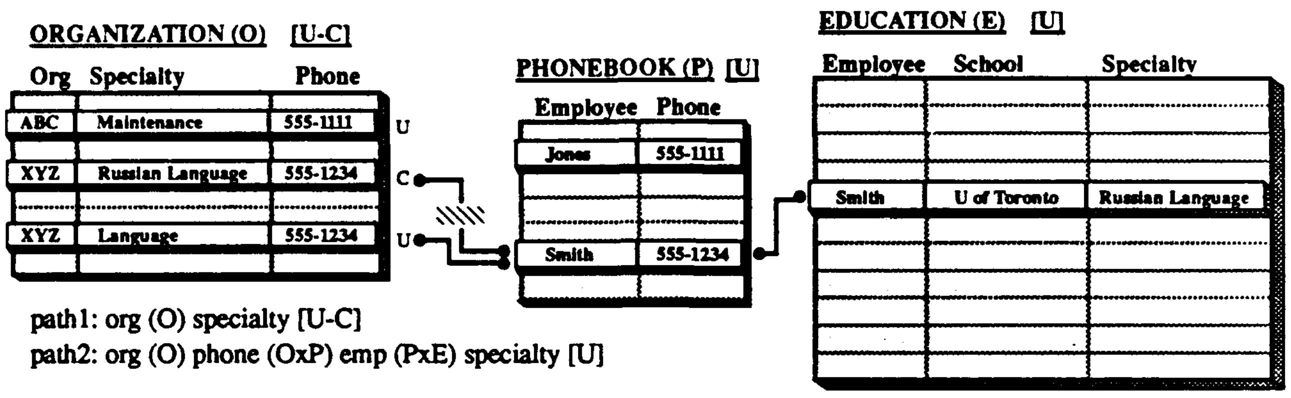

ORGANIZATION (O) [U-C]
Org Specialty Phone
ABC Maintenance 555-1111 U
XYZ Russian Language 555-1234 C
XYZ Language 555-1234 U
PHONEBOOK (P) [U]
Employee Phone
Jones 555-1111
Smith 555-1234
EDUCATION (E) [U]
Employee School Specialty
Smith U of Toronto Russian Language
path1: org (O) specialty [U-C]
path2: org (O) phone (OxP) emp (PxE) specialty [U]

Figure 4

potentially classified path between org and specialty through the path org (O) specialty, and a potentially unclassified path between org and specialty through the path org (O) phone (OxP) emp (PxE) specialty. It was not difficult to create an example to substantiate this, and it is shown in figure 4.

Figure 4 shows that the polyinstantiation of tuple XYZ at the unclassified level introduces the inference channel. In short, it gives the "hooks" necessary for an unclassified user to make the inference (i.e.: O.phone). The user querying the system sees that XYZ's specialty is "Language," but more specifically it is possible that it's specific specialty is "Russian Language."[1]

If the polyinstantiated row were meant to provide a more anonymous cover, the inference would be even more evident. Consider that instead of "Language" the cover was "Maintenance." Now an unclassified user could query the system and see that XYZ's specialty is "Maintenance" but an employee's specialty [who works for XYZ] is "Russian Language." The user is likely to deduce not only the true specialty of XYZ, but also the fact that XYZ considers this information sensitive enough to h:de.

1. If a substantial number of employees have the specialty of "Russian Language" then the inference becomes stronger.

Database Design

The problem here stems from database design, and the decision to use the correct phone number for the organization's unclassified tuple.

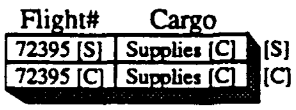
The user maintaining XYZ knows that it is the association between org and specialty that is potentially sensitive. He "knows" that the association between org and phone is not sensitive.

But this user isn't aware of the entire design of the database. In fact, he doesn't know that the relations Phonebook and Education exist. It is easy to see in this example that supplying the correct phone number in conjunction with the design of this database is unsecure. It is not likely that the average user will be able to identify similar security holes given a limited view of a large database system. Even if a single user were required to oversee the entire database schema, they could not analyze all paths without an automated tool.

The purpose of an inference tool is to identify poor database design and make the user aware of potential inference paths. Ideally, an inference engine should prevent the user from creating a table that would lend itself to an illegal inference path. Alternatively, the inference engine could monitor the *contents* of the database based on design ·ime analysis. In this case the engine would allow the existence of a poorly designed database, but would not allow the insertion of data that would allow a potential inference path to become an actual inference path.

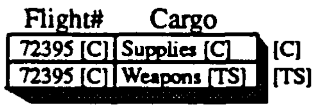
Types of Polyinstantiation

The primary focus on polyinstantiation in recent years has been the polyinstantiation of primary key values (fig. 5), and polyinstantiation of elements having the same primary key (fig. 6).



Flight#	Cargo	
72395 [S]	Supplies [C]	[S]
72395 [C]	Supplies [C]	[C]

Figure 5

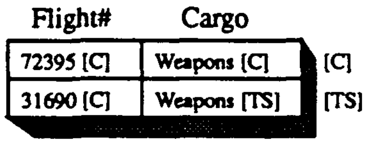


Flight#	Cargo	
72395 [C]	Supplies [C]	[C]
72395 [C]	Weapons [TS]	[TS]

Figure 6

These notions were first introduced by SRI during their SeaView project [4,5]. It is likely that these two classes of polyinstantiation have received so much attention because they complicate the SeaView implementation (namely, the reconstruction of the relational tables).

A more general type of polyinstantiation is simply data polyinstantiation (fig. 7), which encompasses the two previously mentioned.

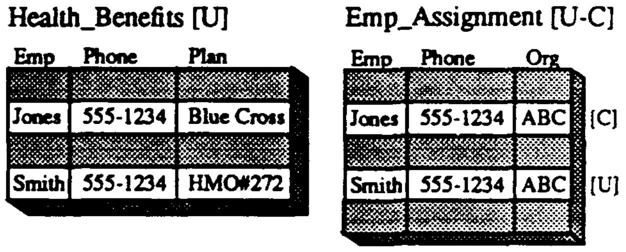


Flight#	Cargo	
72395 [C]	Weapons [C]	[C]
31690 [C]	Weapons [TS]	[TS]

Figure 7

Data polyinstantiation exists when two or more instances of an attribute share the same value but at different security levels; the attribute need not be a primary key. This notion of polyinstantiation does not fall within the definition presented by SRI, but it is an important factor when considering inference.

To see how data polyinstantiation is associated with inference, consider the example in figure 8. In this example there are two tables, each maintained by a different administrator having little or no knowledge of the overall database design. The Health_Benefits table maintains an employee's insurance carrier, the employee's phone number, and other related information. The Emp_Assignment table maintains data on an employee's current assignment. The fact that an employee is working in organization ABC is not normally considered sensitive. Jones, however, was just assigned to organization ABC and he is expected to make several trips out of the country. To maintain a low profile it is decided the relationship between Jones and ABC should be considered confidential, and is reflected in the Emp_Assignment table.



Health_Benefits [U]

Emp	Phone	Plan
Jones	555-1234	Blue Cross
Smith	555-1234	HMO#272

Emp_Assignment [U-C]

Emp	Phone	Org	
Jones	555-1234	ABC	[C]
Smith	555-1234	ABC	[U]

Figure 8

An unclassified user performing the following direct query on Emp_Assignment will not be given Jones' organization:

```
Select EA.org
From  Emp_Assignment EA
Where  EA.emp = "Jones";
```

The next query uses the fact that the phone numbers of Jones and Smith are the same (and polyinstantiated in Emp_Assignment):

```
Select EA.org
From  Health_Benefits HB,
        Emp_Assignment EA
Where  HB.phone = EA.phone
  and  HB.emp = "Jones";
```

This query will succeed in disclosing Jones' organization to the unclassified user. In this case, one solution is to classify all phone numbers in Emp_Assignment which co-exist at the classified level. In a record level environment such as this, Smith's entire Emp_Assignment record would be classified. Other solutions exist, such as classifying Jones' Health_Benefits record. Each of these solutions can only be enforced at run-time; they do not solve the underlying design flaw. To resolve the underlying design flaw the database must be redesigned (i.e. strictly classify Emp_Assignment as confidential).

Defining Inference Through Polyinstantiation

The design flaw allowing polyinstantiation to open an inference channel can be characterized by a relation whose attribute leads to an external attribute co-existing in the original relation.

Formally stated, potential inference through polyinstantiation (I_p) in a database design is defined[1,2]:

$$\begin{aligned} I_p(a_i,a_j) = [\text{TRUE} \mid \exists\ & p_1,p_2,a_k,r,A_1,A_2,R_1,R_2,t \\ & [p_1 = a_i\ (r)\ a_j = P(a_i,a_j,A_1,R_1) \\ & \wedge\ p_2 = P(a_i,a_j,A_2,R_2) \\ & \wedge\ a_i \neq a_j \neq a_k \\ & \wedge\ a_i\ (r)\ a_k \in p_2 \\ & \wedge\ \neg(L(p_1,t) \leq L(p_2,t))]] \end{aligned}$$

Looking back at our example in figure 4, we see that *specialty* (a_j) is the attribute that is both external and local to Organization. *Phone* (a_k) is the attribute or "hook" that can be used in a path leading to *specialty* outside of Organization. Using our definition, we see that a potential inference does exist. The values used to substantiate this are shown below.

a_i = org, a_j = specialty, a_k = phone

A_1 = {org, specialty}

A_2 = {org, phone, emp, specialty}

R_1 = {O}

R_2 = {O, E, P}

p1 = org (O) specialty

p2 = org (O) phone (OxE) emp (ExP) specialty

org (O) phone $\in p_2$

$\neg(L(p_1{=}C) \leq L(p_2{=}U))$

$\therefore\ I_p(\text{org,specialty}) = \text{TRUE}$

Notice that we assign path classifications to suit our needs. The only constraint is that the levels assigned are consistent with the range of possible values. For example, Organization can be assigned either unclassified (U) or confidential (C) security levels however Phonebook is strictly unclassified. The fact that there exists a potentially classified path p_1 and a potentially unclassified path p_2 is a necessary ingredient to show the design is inherently flawed and could <u>potentially</u> breed inference.

The actual tuple values shown in figure 4 are not used when determining the soundness of the design. They are used here to illustrate how a poor design could lead to an inference path, via specific database instance.

Where it is infeasible to eliminate potential inference paths, run-time analysis would monitor the contents of the database based on design-time analysis. Run-time analysis would substantiate when a potential inference path becomes an actual inference path.

1. In this context, $\in$ is used to denote a subpath.
i.e. org (O) phone $\in$ org (O) phone (OxE) emp (ExP) specialty
2. See Appendix B for definitions of $P(a_0,a_n,A,R)$ and $L(p)$

Resolving Inference

Preventing inference has the same implications as preventing polyinstantiation. The act of prevention *itself* can be used to create a covert channel. In fact, our options are even more restricted. One method of resolving polyinstantiation is to delete the more highly classified tuple [3]. This is not a solution to inference, since by definition the user accesses only unclassified data to make the inference. Removing the classified table would only limit your knowledge of what classified information the user might infer. Preventing inference requires that some portion of the unclassified path be deleted or upgraded to a higher classification. Some classes of information are inherently difficult to protect. It is important that system security is not based on the protection of these classes of information.

There is a period of time between the creation of an inference channel and the use of that channel to compromise unauthorized information. For design-time analysis that period is the time between creation of the relation and the query of information inserted into that relation. Although a potential inference can be detected at design-time, the inference channel cannot become active until data is actually inserted into the relation. For run-time analysis the period is the time between insertion of data and subsequent query of that information. It is during this period of time that measures must be taken to ensure unauthorized information is not compromised through inference.

Inference prevention does leave us with the problem of covert channels. In an extreme case, the covert channel can be limited to one bit of information by denying further access to the system. Other options exist and must be identified in the trusted facility manual. These options are themselves a separate issue and are not addressed in this paper.

Summary

Polyinstantiation, although useful in preventing a class of covert channels, has the potential to open a new class of inference channels. This paper does not advocate the use nor prevention of polyinstantiation; it identifies a weakness of it. In short, polyinstantiation:

- Complicates inference path analysis
- Must be taken into consideration at database design-time and possibly at run-time
- May provide an inference channel especially when used as a cover story mechanism in conjunction with a poorly designed database

These factors must be taken into consideration when deciding whether polyinstantiation should be allowed, and [if it is] what measures need to be taken to ensure inference channels are identified and eliminated[1].

While creating cover stories, special care must be taken when polyinstantiating data. In some cases it may not be possible to provide a cover for each individual attribute. When not essential for the plausibility of a cover, data should not be polyinstantiated. When it is essential, the schema should be examined to ensure that the polyinstantiation does not compromise classified information.

As shown in SWORD [8], polyinstantiation of key values can be prevented. Additionally it is shown that cover stories can be implemented using column or element level labeling without the need for polyinstantiating primary key values. Their methodology, however, does not address polyinstantiation of non-key values and their affect on inference channels.

An inference tool is necessary to make the user aware of inherently unsecure database

1. or *reduced*, depending on the target security level

design schemas. With a sufficiently large database, an administrator would unable to perform path analysis without an automated tool.

By doing analysis at design-time, overhead at query time is eliminated. Where detection of potential inference is too restrictive, run-time analysis could monitor the *contents* of the database based on design-time analysis. In this case the tool would allow the existence of a poorly designed database, but would flag the insertion of data that would allow a potential inference path to become an actual inference path. Run-time analysis leads to higher overhead during update, but makes design capability more flexible.

We have formally defined potential inference through polyinstantiation. *Implementing* an inference tool, however, is a hard problem and an area for further research. Brute-force solutions will undoubtedly lose out to time constraints; heuristic techniques or partial solutions may provide us with the best hope of a meaningful answer to the inference problem.

Inference is not a monolithic problem, and cannot be treated as such. Appendix B formally defines inference through secondary path analysis. The fact that it does not detect inference through polyinstantiation is not an error, because its sole purpose is to define a specific class of inference; and it does. The ultimate goal of this work is to develop a comprehensive inference policy capable of being implemented. To accomplish this, it is essential to recognize each class of inference individually, and propose solutions to each of these specific classes. Only then will it be possible to bring each of these individual solutions together within the structure of a single, comprehensive inference policy.

References

[1] Binns L.J. "Inference through Secondary Path Analysis," Unpublished draft, June 1991.

[2] Hinke T.H. "Inference Aggregation Detection in Database Management Systems," *Proceedings of the 1988 IEEE Symposium on Security and Privacy*, Oakland, CA, April 1988.

[3] Jajodia S. and Sandhu R. "Polyinstantiation Without Sacrificing Integrity o. Availability," *Proceedings of the Fourth RADC Database Security Workshop*, April 1991.

[4] Lunt T.F. "The True Meaning of Polyinstantiation: Proposal for an Operational Semantics for a Multilevel Relational Database System," *Proceedings of the Third RADC Database Security Workshop*, June 1990.

[5] Lunt T.F. "Polyinstantiation: an Inevitable Part of a Multilevel World," *Proceedings of the Fourth Workshop on the Foundations of Computer Security*, Franconia, New Hampshire, June 1991.

[6] Morgenstern M. "Controlling Logical Inference in Multilevel Database Systems," *Proceedings of the 1988 IEEE Symposium on Security and Privacy*, Oakland, CA, April 1988.

[7] Thuraisingham B. "Handling Association-Based Constraints During Database Design," *Proceedings of the Fourth RADC Database Security Workshcp*, April 1991.

[8] Wood A.W. "SWORD: A Truthful Multilevel Secure Database," *ESORICS 90*, Toulouse, France, October 1990.

Appendix A
Mock Classification Guideline

- An organization's name alone is not classified
- An organization's specialty is not classified unless stated otherwise
- The relationship between organization and specialty is confidential if its specialty is one of the following:
    - Russian Language
    - Metalinguistics
    - Optical Fiber Transmission
    - Civil Engineering
- If the specialty falls within these classified areas, the following unclassified specialties are to be used when referencing that organization to an unclassified user:

Classified Specialty	Unclassified Specialty Description
Russian Language Metalinguistics	Language
Optical Fiber Transmission Civil Engineering	Engineering

- To provide consistency for the unclassified user, any record which relates organization and specialty at the confidential level must be polyinstantiated at the unclassified level. The record at the unclassified level will reflect the organization's unclassified specialty.

Appendix B

Definition of Inference Through Secondary Path Analysis

Following is a formal definition[1] of potential inference using a non-weighted, non-directed graph This definition does not consider the polyinstantiation of tuples at different security levels.

<u>A path of length (n=1) is defined as:</u>

$P(a_0,a_1,A,R) = [a_0\ (r_1)\ a_1 \mid a_0,a_1 \in A$
$\wedge\ a_0 \neq a_1$
$\wedge\ r_1 \in R$
$\wedge\ a_0,a_1 \in r_1$
$\wedge$ Cardinality(A) = 2
$\wedge$ Cardinality(R) = 1]

<u>A path of length (n > 1) is defines as:</u>

$P(a_0,a_n,A,R) = [P(a_0,a_{n-1},A-a_n,R-r_n)\ (r_{n-1} \times r_n)\ a_n \mid$
$a_0,a_{n-1},a_n \in A$
$\wedge\ a_0 \neq a_{n-1} \neq a_n$
$\wedge\ r_{n-1},r_n \in R$
$\wedge\ r_{n-1} \neq r_n$
$\wedge\ a_{n-1},a_n \in r_n$
$\wedge\ a_{n-1} \in r_{n-1}$
$\wedge\ P(a_0,a_{n-1},A-a_n,R-r_n)]$

Where A is a set of n+1 attributes
and R is a set of n relations

<u>The level of a path is defined by the relations used in traversal:</u>

$L(p,t) = [L_h(R,t) \cup L_c(R,t) \mid \exists\ a_0,a_n,A\ (p = P(a_0,a_n,A,R) \wedge t \in \text{time}]$

Where

$L_h(\text{nil},t) = U$
$L_h(R,t) = [\text{Level}(r,t) \mid r \in R \wedge \text{Level}(r,t) \geq L_h(R-r,t)]$

$L_c(\text{nil},t) = \text{nil}$
$L_c(R,t) = [\text{Compartment}(r,t) \cup L_c(R-r,t) \mid r \in R]$

Level(r,t) = Hierarchical security level associated with relation *r* at time *t*
Compartment(r,t) = Non-hierarchical compartment associated with relation *r* at time *t*

R = set of relations

{U, C, S, TS} = Hierarchical security levels, and U < C < S < TS

1. This definition is taken from a working paper [1] not yet submitted for publication

Levels (because of compartments) are only partially ordered.
The $\leq$ relationship is defined as:

$$
\begin{aligned}
L(p_1,t) \leq L(p_2,t) \Leftrightarrow \exists\, & a_i,a_j,a_k,a_l,A_1,A_2,R_1,R_2 \\
& [p_1 = P(a_i,a_j,A_1,R_1) \\
& \wedge p_2 = P(a_k,a_l,A_2,R_2) \\
& \wedge L_h(R_1,t) \leq L_h(R_2,t) \\
& \wedge L_c(R_1,t) \subseteq L_c(R_2,t)]
\end{aligned}
$$

An illegal inference through secondary path analysis is defined as:

$$
\begin{aligned}
I(a_0,a_n) = [\mathrm{TRUE} \mid \exists\, & A_1, A_2, R_1, R_2, p_1, p_2, t \\
& [p_1 = P(a_0,a_n,A_1,R_1) \wedge p_2 = P(a_0,a_n,A_2,R_2) \\
& \wedge L(p_1,t) \neq L(p_2,t) \\
& \wedge \forall r_1,r_2\, [r_1,r_2 \in (R_1 \cap R_2) \wedge r_1 = r_2 \\
& \qquad \rightarrow L_h(r_1,t) = L_h(r_2,t) \ \wedge L_c(r_1,t) = L_c(r_2,t)]]]
\end{aligned}
$$

Note:

1) The same relation can be used in both paths when proving inference

2) If the same relation exists in both paths, it must be assigned the same security level in each for the purpose of determining inference; i.e. because a table is multilevel in and of itself cannot be cited as the cause of inference.

3) This definition was written for a multilevel environment with tuple level labeling. Slight modification of the definition is required to suit column level or element level granularity.

SECTION 4

MLS DBMS ENGINEERING

The papers in this section of the report address issues and solutions for building database management systems that enforce mandatory security. Ira Greenberg's paper "Should Serializability be Enforced in Multilevel Database Systems?" explores the goals of concurrency control mechanisms and suggests new correctness criteria for multilevel concurrency control algorithms. The evaluation of database management systems at high levels of assurance using the Trusted Database Interpretation (TDI) does not appear to be feasible in the near term; the paper by Irvine, Shell, and Thompson addresses this problem by exploring the use of the Trusted Network Interpretation (TNI) as a basis for evaluation. The paper describes the concepts of TCB subsets and balanced assurance and discusses how those concepts relate to evaluations based on the TNI. In his paper "Evaluation by Parts of Trusted Database Management Systems," Ravi Sandhu describes how some of the conservative assumptions of the TDI might be relaxed based on specific DBMS and operating system architectures. He concludes that additional research is needed to determine the specific conditions where the architecture has an impact on the feasibility of evaluation by parts. Dick O'Brien looks at the problems of correctly enforcing mandatory access controls with respect to the data dictionary relations in a relational database. His paper "Metadata Issues in a MLS DBMS" identifies several avenues of disclosure and denial of service in the management of metadata and proposes some approaches to addressing them.

Should Serializability be Enforced in Multilevel Database Systems?

Ira B. Greenberg
SRI International
333 Ravenswood Avenue
Menlo Park, CA 94025

Abstract

Security requirements in multilevel database systems interact with concurrency control properties and algorithms. First, the multilevel database models being studied are described. The known secure serializable algorithms are then presented, and their drawbacks are discussed. It is then argued that the additional semantic knowledge available about integrity constraints in a multilevel database makes it possible to devise nonserializable algorithms that preserve correctness and exhibit improved behavior. A new correctness criteria called multilevel correctness is then proposed, and it is used to examine the nonserializable concurrency control algorithm employed by Trusted Oracle.

1 Introduction

Concurrency control is a fundamental component of a database system. Its purpose is to manage the concurrent execution of transactions, so that their operations do not interfere and cause errors. A particular approach to concurrency control will support certain properties about transaction execution and the correctness of database states. For example, the standard correctness criteria for concurrency control is serializability, which guarantees that the concurrent execution of a collection of transactions will have the same effect as some serial execution of the transactions. A variety of algorithms can be used to implement the same property, although each algorithm may have different performance characteristics and be more suited for particular job mixes.

Concurrency control in a multilevel database system interacts with the security model being enforced by the database system. We have been studying the effect of a high assurance security model on concurrency control to determine what properties can be provided. In the security model we are considering, each subject is single level, each transaction is single level, a subject can read only from dominated access classes, and a subject can write only at its access class. The subjects include those that implement the database system. Mandatory security is enforced by the operating system's trusted computing base (TCB).

Our work has shown that it is possible to support serializability for a multilevel database system with the stated security model. However, the two known serializable techniques both have drawbacks. The problem with the first algorithm, which is based on optimistic concurrency control, is that the execution of low level transactions can cause the execution of high level transactions to be delayed for an arbitrary length of time. The problem with the second algorithm, which is based on multiversion timestamp ordering, is that high level transactions can be forced to read arbitrarily old data values as a result of the execution of low level transactions.

Serializability is typically selected as the correctness criteria for concurrency control because (1) it correctly provides the greatest amount of concurrency with the transaction information that is available, (2) it is well understood, and (3) efficient algorithms for serializable execution exist. In a multilevel database system, however, additional semantic knowledge is available, which makes it possible to employ nonserializable concurrency control algorithms. In addition to providing better performance, we believe that these nonserializable algorithms are less severely affected by the security model we are studying. Consequently, we believe that maintaining serializability in a multilevel database system is overkill, and that nonserializability should be used.

This paper is organized as follows. The first section describes the abstract database models we are using, and the second section describes the security model we are trying to enforce. This is followed by a discussion of correctness criteria for the execution of concurrent transactions, including serializability. We then outline the two known serializable concurrency control algorithms for the models being studied. The next section describes how the additional semantic knowledge in a multilevel database system can be used to devise nonserializable algorithms. We then propose a set of properties for nonserializable correctness control in a multilevel database system. These properties are then used to examine the nonserializable concurrency control algorithm implemented by the Trusted Oracle multilevel database system. The paper concludes with a summary.

2 Database Model

We will use the abstract architectural models for a database system described by Bernstein et al. [1] as the basis for our discussion of database algorithms and properties. These models include the database, database operations, and the modules comprising the database system that manages the database. In this section, we will describe two models for a single-site database system. The first, the single version model, stores only the latest version of each data item, while the second, the multiversion model, stores several versions for each data item. This paper will be presented in terms of the single-version model, except where noted.

2.1 Single-Version Database Model

A single-version database is a set of named data items, each with a value. The data items will be represented by lower case letters, such as x and y. The values of the data items at a given time form the database state.

A database system is a collection of software modules that manages the database. It executes data operations on the database on behalf of users, and returns results to the users. The database system is also responsible for performing common service functions for the users, such as executing operations efficiently, maintaining the integrity of the database when failures occur, and so forth.

The basic database operations are the read function, which returns the value of a data item, and the write function, which changes the value of a data item. The data operations are executed atomically, which means that they appear to execute sequentially and either execute in their entirety or not at all. If they actually execute concurrently, that fact is not detectable outside the database system.

When database operations are executed, they are organized into larger operations called transactions. Each database operation must execute as part of some transaction. Transactions, which we will denote by T_1, T_2, and so on, are considered to be the unit of database work. There are three transaction operations: start, commit, and abort. The start operation indicates that a transaction has begun and that the following database operations should be grouped together. A transaction is ended by an abort or a commit operation. The abort operation indicates that a transaction has failed for some reason, that its execution should be terminated, and that any effects caused by its execution should be removed. The commit operation indicates that a transaction has completed successfully, and that its results should be permanently recorded.

The standard transaction properties are atomicity, serializability, and persistence. Here, atomicity means that a transaction will either be executed in its entirety, or not at all. Usually, it is not possible to access the partial results of a transaction. Serializability guarantees that the interleaved execution of a collection of transactions will have the same effect as some serial execution of the transactions. Persistence means that the effects of a transaction will be permanent once the transaction is completed and committed.

The abstract model of a centralized database system is shown in Figure 1, which is taken from Bernstein et al. [1]. The model consists of four modules: the transaction manager, scheduler, recovery manager, and cache manager. At a higher level of abstraction, the recovery manager and cache manager can be viewed together as a single module, the data manager, that directly accesses the database.

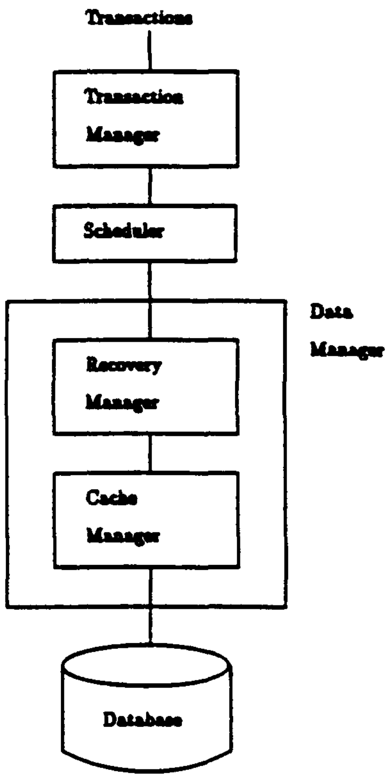


Figure 1: Abstract model of a centralized database system

Users interact with the database through transactions, which are received and managed by the transaction manager. Among other tasks, the transaction manager passes the database and transaction operations in the transactions to the scheduler. The scheduler is responsible for controlling the concurrent execution of transactions. It accomplishes this task by specifying the order in which the operations are executed by the data manager. Its goals are to maintain the integrity of the database by ensuring that transactions are executed serializably and recoverably, to permit high performance by allowing a high degree of interleaving, and to maintain other transaction properties.

The data manager is responsible for managing the database storage, both on disk and in memory. Read and write operations are sent from the scheduler to the data manager. The recovery manager implements these operations in a reliable manner. The cache manager controls the movement of data between memory and disk.

2.2 Multiversion Database Model

In the standard database model, where there is only one version of each data item, all transactions compete for the most recent values of the data items. By expanding the database model to allow multiple versions for each data item, conflicts between transactions can be reduced and performance can be improved.

A multiversion database maintains multiple versions of each data item and maps the execution of transactions on data items to executions on particular versions of the data items. Transaction operations are written so that they refer to data items without specifying a version. The concurrency control mechanism implemented by the scheduler automatically translates these operations so that they execute on the appropriate versions of the data items. A user is not expected to know that there are multiple versions of data items, and is not allowed to explicitly reference any versions.

The standard definition of serializability does not work for multiversion databases because it does not account for the different versions of data items. The analogous correctness criterion that was defined for multiversion databases is called multiversion serializability. The interleaved execution of a collection of transactions on a multiversion database is *multiversion serializable* if it is equivalent to some serial execution of the transactions on a database with only single versions of the data items. Once again, the executions are mapped back to a known situation, and this execution will be correct because the serial execution of transactions on a single-version database is correct.

3 Security Model

Various security models have been proposed for multilevel database systems. We have selected an abstract model similar to the SeaView model [8, 2] because we believe it affords the greatest security assurance. In this study, we are focusing on mandatory security and are not considering discretionary security. Mandatory security is the ability to control direct and indirect access to data according to subject requesting access.

In this multilevel database system, the data items are the storage objects, and they are each assigned an access class. The processes that act on behalf of users, including the processes that make up the database system, are the subjects, and they are each assigned an access class. Each access class may consist of a hierarchical level, such as Secret or Confidential, and a set of nonhierarchical categories, such as NATO or Nuclear.

The set of all possible access classes is structured as a lattice by a partial ordering relation called "dominates." One access class strictly dominates another if its hierarchical level is greater than the other's hierarchical level and its categories are a superset of the other's categories, or if their hierarchical levels are the same and its categories are a strict superset of the other's categories. An access class dominates another if it strictly dominates it or if they are identical. If neither access class dominates the other, they are called incomparable.

The following two rules are used to protect data items from unauthorized disclosure and contamination when they are accessed by subjects.

- **Simple security property**: A subject can read only data whose access class is dominated by the subject's access class.

- **$\star$-property**: A subject can write only data whose access class equals the subject's access class.

To use the database, a user must log in to the database system at one of the access classes encompassed by the user's clearance. The user will access the database through a subject whose access class will be set to the access class at which the user logged in. The subject will access the database by sending a transaction to the transaction manager. Each transaction will have a single access class equal to that of the subject that issued it. The operations that comprise a transaction will in turn be assigned the access class of the transaction.

Because subjects are single level, including the components of the database system, the operating system must be able to instantiate a version of the database system with the access class of any subject that needs to access the database. A version of the database system may be composed of many subjects, all of which will be single level and at the same access class.

We assume that the system is designed with a TCB subsets approach [11, 9]. In this approach, an underlying mandatory security kernel is used to enforce mandatory security for the entire system. Each access by a subject is intercepted by the TCB and validated against the two access rules described above. The database system executes above the TCB and relies on the TCB to enforce mandatory security. The single-level subjects in the database system do not require new security proofs because they cannot introduce any compromise to mandatory security.

Many of the properties provided by a database system, such as those related to concurrency control and recovery, are global properties that are applied to all transactions or data items. Supporting a global property for a multilevel database without violating security requirements is impossible or difficult, depending on the property, because all the database modules are implemented by single-level subjects. The simple security property prevents all subjects except those at the highest access class from seeing all of the data items. The $\star$-property prevents a subject from acting at all access classes except its own.

To implement a global property, subjects at multiple access classes must cooperate and coordinate their actions. By employing the same rules and assuming that subjects at other access classes will behave in a certain way, the collective effect of their single-level algorithms can have the same effect as the desired global algorithm. This is the strategy that will be followed for the properties that will be examined. Sometimes it is impossible to implement a global property with a collection of single-level algorithms. At other times, the global property can be achieved, but performance or other characteristics are adversely affected.

4 Correctness and Serializability

A database state is considered to be correct if a collection of integrity constraints is satisfied for the contents of the database. Typically, each transaction is assumed to be correctly written so that its execution will maintain the integrity constraints. If this is true, then the execution of a single transaction will transform one correct database state to another correct database state. By induction, the serial execution of a collection of transactions will preserve the correctness of the database.

When transactions execute in parallel, their operations can be interleaved. These interleaved operations can interfere and cause incorrect results. Forcing the transactions to execute serially will result in correctness, but it will also lead to poor performance. In general, to show that the database states produced by transactions are correct, it is necessary to show that all of the constraints are satisfied for each new state. This can be prohibitively expensive, and often many of the constraints are not even known by the database system.

Concurrency control is implemented by the database system component called the scheduler. The purpose of the scheduler is to manage the interleaving of transaction operations so that the correctness constraints are satisfied and a high level of performance is achieved. The scheduler's input consists of a stream of requests for the execution of the transaction operations. It can grant, delay, or abort each of the requests. The scheduler's output for the transaction operations is an equivalent schedule, which is to be executed by the data manager.

The correctness and performance of a scheduler's output depends on the information available to the scheduler about the transactions and the database. As discussed by Kung and Papadimitriou [6], the information used by a scheduler is the minimum knowledge about the database and the transactions that the scheduler requires to function correctly. Information that would be useful to the scheduler includes syntactic information about the transactions, semantic information about the meaning of the data and the operations performed, and information about the integrity constraints that the data must satisfy.

4.1 Serializability

The standard correctness criterion for the execution of concurrent transactions is called serializability. The interleaved execution of a collection of transactions is serializable if its result is equivalent to that of some serial execution of the transactions. The execution of a serializable transaction history is guaranteed to be correct because serial executions are correct.

Serializability has been studied extensively, and numerous efficient algorithms are known. An important advantage of serializability is that it is relatively easy to produce serializable histories. Rather than checking a transaction history after it is generated, the scheduler can employ concurrency control algorithms that are guaranteed to produce only serializable histories. This allows greater performance than serial transaction execution, and is less expensive to accomplish than checking correctness constraints.

Kung and Papadimitriou have shown that, when only syntactic information is available to the scheduler, serializability provides the maximum amount of concurrency [6]. This is often the case in real database systems, with semantic and integrity constraint information either unavailable, difficult to specify, or difficult to exploit. If additional information was available, however, it would be possible to devise concurrency control algorithms that were correct and nonserializable, and that gave better performance than serializable algorithms. We will discuss the use of nonserializable correctness criteria in a multilevel database system in Sections 6 and 7.

4.2 Security and Serializability

In a multilevel database system, there is the additional problem of maintaining correctness for concurrently executing transactions without violating security requirements. The fact that all subjects must be single level precludes the existence of a global scheduler. Instead, there is one single-level scheduler for each access class. The goal for the group of schedulers is to collectively produce a serializable global transaction history—that is, a history equivalent to some serial transaction history that could be produced by a hypothetical global scheduler. Each single-level scheduler is assigned an access class by the TCB when it is created. Each schedules only requests from transactions at its access class, but it must do so in a way that coordinates with the actions of the other schedulers.

This approach cannot compromise mandatory security because the mandatory security policy is enforced by an underlying TCB that is controlling accesses by subjects to storage objects. However, it may be difficult to create correct schedules, because all the schedulers must cooperate to produce a correct global schedule without full communication. Because single-level schedulers cannot write down or read up, communication can only be one way, with a scheduler able to obtain information only from dominated schedulers.

The use of multiple single-level schedulers is illustrated in Figure 2. If transactions a and c are submitted by Unclassified subjects and transaction b is submitted by a Classified subject, then the global input would be broken down as shown. Each single-level scheduler would then produce a single-level output schedule for its transactions; these single-level schedules would collectively form the global output schedule. Note that the global input and output schedules are not explicitly formed or represented in the system. They are the implicit sequence of transaction operations that would be seen by an omniscient observer. The order of operations in the global output schedule is the order in which the operations were executed by the single-level schedulers. The combined effect of the single-level schedules will be serializable if the global output schedule is equivalent to some serial execution of the three transactions.

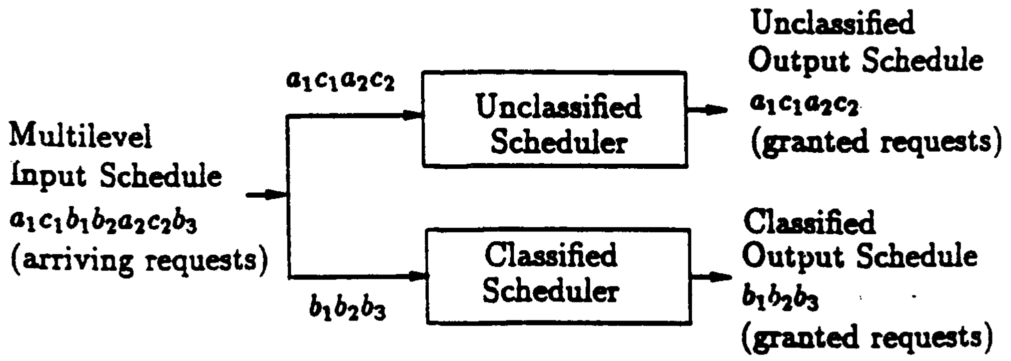


Figure 2: Schedule generation by multiple single-level schedulers

5 Secure Serializable Concurrency Control Algorithms

Only a few secure serializable concurrency control algorithms have been developed for a multilevel database system based on the database and security models we have been studying. These algorithms use two basic approaches for realizing serializability. The first approach is to delay the execution of a transaction until it does not conflict with transactions at lower levels. The second approach is to execute a transaction using old versions of lower-level data items so that it will not conflict with transactions at lower levels. In both cases, a transaction is responsible for avoiding or recovering from conflicts with lower-level transactions.

In this section, we will briefly review examples of these approaches, and discuss their drawbacks. The first algorithm is called modified optimistic concurrency control [4, 3], and is designed for single-version databases. It is based on Kung and Robinson's optimistic concurrency control approach [7], and works by delaying the execution of transactions when necessary. The second algorithm is called MVTO-SS [4, 10], and is designed for multiversion databases. It is based on the multiversion timestamp ordering (MVTO) approach [1], and works by allowing transactions to use old data values when necessary.

5.1 Modified Optimistic Concurrency Control

Unlike other approaches, which prevent conflicts from occurring, optimistic concurrency control allows a transaction to complete execution and then validates the transaction result when the transaction is ready to commit. If the transaction has conflicted with another transaction and has used potentially inconsistent data, it is rolled back and restarted. This method is called "optimistic" because its efficiency relies on a low conflict rate and infrequent rollbacks.

For optimistic concurrency control, transactions are required to consist of three phases: a read phase, a validation phase, and a (possibly empty) write phase. During the read phase, a transaction obtains private copies of all relevant data, and makes all updates on these private copies. The validation phase certifies that the transaction does not conflict with transactions that previously committed or are currently committing. If the transaction is validated, the private copies are made public during the write phase.

Modified optimistic concurrency control is based on optimistic concurrency control with serial validation as described by Kung and Robinson [7]. In the new algorithm, a standard version of optimistic concurrency control is run among the transactions of a given access class to prevent conflicts between the transactions at that access class.

A modified version of optimistic concurrency control is used to synchronize transactions from different access classes. The enforcement of mandatory security by the TCB makes it impossible for a transaction's result to depend on information written by a higher-level transaction. Therefore, a transaction must be validated only against transactions from dominated access classes. Consequently, a single-level scheduler can always read the information it needs to validate a transaction executed at the scheduler's access class, and the database system can always securely roll back every transaction that needs to be rolled back. A small amount of information must be logged for each transaction so that the single-level schedulers can coordinate their actions. Additional details can be found in [4, 3].

5.2 Modified Multiversion Concurrency Control

The basic MVTO algorithm [1] maintains multiple versions of data items to reduce conflicts. The types of conflicts that can still occur are resolved by immediately aborting the transaction that causes the conflict. MVTO cannot be used for a multilevel database system because mandatory security will prevent certain aborts from occurring, thereby leading to incorrect results. We will now describe the basic MVTO algorithm and outline the modifications necessary to create a secure version of the algorithm.

The basic MVTO algorithm works as follows. A transaction is assigned a unique timestamp when it begins executing, and this timestamp is associated with all read and write operations in the transaction. Essentially, each transaction executes instantaneously at a unique virtual time. Each data item can have many versions, and each version has a read timestamp and a write timestamp associated with it. When a data item is written, a new version of the data item is created, and its write timestamp is set to the timestamp of the write operation. When a data item is read, the database system reads the version of the data item with the most recent write timestamp earlier than the timestamp of the read operation. That version's read timestamp is then set to the more recent of its current value and the read operation's timestamp. A write will be rejected, and the transaction that issued it will be aborted, if the write will invalidate the result previously returned by a read. More precisely, a write will be rejected if its timestamp falls between the write and read timestamp for a version of the data item being written. Note that writes from different transactions cannot conflict because the writes have different timestamps.

MVTO would provide serializable concurrency control for a multilevel database system if it were possible to *a priori* guarantee that a transaction would never invalidate a read executed by a transaction from a higher access class. Keefe and Tsai [5] used this insight to develop a modified version of the MVTO algorithm that would provide secure serializable concurrency control. However, their algorithm required a trusted subject to perform scheduling, which is not permitted by our security model. MVTO-SS was created by modifying Keefe and Tsai's algorithm to remove the need for a trusted subject. (SS stands for single-level scheduler.)

The basic idea behind the secure versions of MVTO is that it will sometimes be necessary to assign a new transaction an execution timestamp that is earlier than the current timestamp. This effectively moves the new transaction into the past with respect to all active transactions and some committed values at strictly dominated access classes. To be more precise, when a transaction begins, it is assigned a timestamp that precedes the timestamps of all transactions active at strictly dominated access classes and that follows the timestamps of all transactions at its own access class. This approach to timestamp assignment is the factor that makes it impossible for a transaction to invalidate a read from a higher access class.

MVTO-SS must coordinate the actions of the single-level schedulers so that they cooperate to produce a multiversion serializable schedule. The problem is to correctly assign timestamps to the transactions so that they will execute in the proper order in the global execution history. A labeling scheme is defined that allows the single-level schedulers to independently assign the transactions unique virtual timestamps. In addition, an algorithm is described that allows each single-level scheduler to determine the proper virtual timestamp based on auxiliary information stored at strictly dominated access classes. Additional details can be found in [4, 10].

5.3 Drawbacks

Two techniques have been shown for providing secure, serializable concurrency control, but each has a serious drawback. A modified version of optimistic concurrency control will work, but the execution of transactions that read from strictly dominated access classes can be indefinitely delayed. This performance problem, which is an example of denial of service, is especially significant because the highest-level access classes contain the most sensitive activity and information.

For a multiversion database, a modified version of MVTO, MVTO-SS, could be used, but it has the drawback that transactions can be forced to read arbitrarily old data values from dominated access classes to avoid conflicting with transactions at strictly dominated access classes. Here, the problem is that the quality of the result degrades as the transactions are required to read progressively older data. Once again, it is the work at the highest access classes that will be most seriously affected.

These problems appear to be fundamental consequences of secure serializable concurrency control when the security model requires single-level subjects, and not just deficiencies of the particular algorithms that were described. In the remainder of the paper we examine the benefits of using nonserializable concurrency control techniques.

6 Using Semantic Knowledge to Improve on Serializability

As discussed in Section 4, serializability is only one approach for ensuring correct concurrent execution; the real goal is to ensure that the database integrity constraints are not violated. Based on the work by Kung and Papadimitriou, greater concurrency than is possible with serializability can be achieved only if some semantic knowledge is available about the transaction operations, the database, or the integrity constraints. In the context of a multilevel database, this extra knowledge may also make it possible to achieve correct execution without the kinds of performance and quality drawbacks discussed above.

An examination of the security model we have selected reveals that many kinds of integrity constraints cannot exist between data items from different access classes. We believe that this additional semantic knowledge about integrity constraints can be exploited to develop better concurrency control techniques for the selected class of multilevel database systems.

Some integrity constraints relate the values of two or more data items. Integrity constraints called invariants require the relationship to always hold. Invariants across access classes cannot exist for the selected security model because they cannot be enforced. For example, two data items at different access classes cannot be changed simultaneously (atomically) by a transaction because transactions are single level and can write at only one access class. Also, a transaction changing the value of a data item cannot validate the new value against a data item at a higher or incomparable access class because of the simple security property. Invariants across access classes may be desirable, but they are prohibited by the use of mandatory security and single-level subjects.

A transaction can appear to establish consistency among data items at its access class and data items at strictly dominated access classes, but this is an illusion. The consistency can be broken immediately by a transaction that changes one of the data items at one of the lower access classes. The best that can be done is to have some process at the high access class frequently probe the lower-level data items and reestablish consistency as soon as possible. Not only is this expensive, but the constraint will often be violated anyway.

Another kind of integrity constraint, called an ordering constraint, occurs when a value depends on the fact that some event has occurred, In a database, this dependency is formed when another value, which represents the event, is read from the database. Ordering constraints can exist between data items at different access classes, and must be maintained by the concurrency control mechanism.

Enforcing serializability for the selected type of multilevel database appears to be overkill because it requires establishing global transaction execution orders to preserve nonexistent invariant integrity constraints. While serializable execution is not wrong, it appears to have a high cost in terms of performance or quality of result. We consider the quality of the result to be degraded when transactions are forced to base their results on excessively old versions of data items, ignoring newer values for those items.

7 Nonserializable Correctness

The limitations on integrity constraints make it possible to consider less restrictive correctness properties than those imposed by serializability. An important point about these properties, which should lead to better implementations, is that invariants need to be maintained among transactions only if they are at a single access class. Note that while global serializability is not provided, the contents of the database are still correct because all existing integrity constraints will be satisfied.

The multilevel concurrency control correctness criterion of *multilevel correctness* is defined to consist of the following properties.

- **Single-level serializability.** Arbitrary integrity constraints can exist among the data items at an access class. Because only syntactic information is available within an access class, serializability will have to be used as the correctness criterion for the execution of the transactions at the access class. Any standard serializable concurrency control technique can be used.

- **Single-level read consistency.** Data items within an access class may be related by integrity constraints. When a transaction reads two or more data items related by integrity constraints from a dominated access class, the data items must be read from a consistent collection of values.

- **Progress.** Once a particular version of a data item has been read by a process, any process dependent on the execution of the first process should not read an earlier version of the data item.

The difference between multilevel correctness and serializability is that the value of a data item read from a strictly dominated access class need not be the most recent version of the data item at the time when the transaction performing the read commits. As long as consistent groups of related data items are read, they may be from older versions.

Progress is not strictly a database consistency property. It represents the often unstated assumption that actions or computations move forward in time. In many situations, it is assumed that an action will be able to observe in the current state the effects of previous actions, and should act accordingly with respect to those effects.

In the multilevel database system that has been described, the transactions for each access class are serialized into a sequence. Although invariants are not defined across access classes, transactions do interact and form dependencies across the access classes by the order in which they read and write data items. The purpose of progress is to maintain these computational dependencies.

The values of the set of data items at an access class form an access class state. The execution of the transactions at an access class will cause the access class to move through a series of states. Each state will depend on one or more states from strictly dominated access classes where reads occurred. We will call these the dependent states. When a transaction is executed on an access class state, progress will be satisfied if the concurrency control version function does not select a data item version from a state older than the dependent states. As with serializability, it is desirable to have a simple method for guaranteeing progress, rather than requiring dependencies to be computed and examined.

Another property that is desirable but not necessary is recency. If a transaction can read any of several versions of a data item, it should try to read the most recent version. Presumably this approach will produce the best results because they will be based on the most up-to-date information.

Multilevel correctness is just one way to define correctness for a multilevel database system. Serializability is another valid definition, but it appears to be too constraining. Multilevel correctness is designed to provide a collection of correctness properties that are appropriate for the semantics of a multilevel database system and a group of common processing assumptions. The goal is to allow the performance and the quality of results to be improved by using a weaker correctness criterion that is less restrictive than serializability. Multilevel correctness will provide all of the properties of serializability except the requirement that the values a transaction has read when it commits belong to the most recent versions of the data items. Other correctness criteria can be used that provide a different group of guarantees and may be appropriate for a different group of processing assumptions.

8 Trusted Oracle Concurrency Control

The concurrency control algorithm employed by Trusted Oracle can be used for the multiversion database model and the security model we have selected. Unlike the other known algorithms for these models, the Trusted Oracle algorithm is not multiversion serializable [4]. In this section, we describe the algorithm and examine it with respect to multilevel correctness.

8.1 The Trusted Oracle Concurrency Control Algorithm

The concurrency control algorithm used in Trusted Oracle is a hybrid multiversion algorithm that uses strict two-phase locking (2PL) and MVTO. It is an extension of the concurrency control algorithm used for the single-level Oracle database, which also uses strict 2PL and MVTO. The algorithm works as follows.

Each data item is assigned an access class, and multiple versions are maintained for each data item. Each version of a data item has a timestamp associated with it. A new version of a data item is created for each transaction that writes to the data item. A new version is permanently entered in the database when the transaction that creates it successfully commits, and the timestamp assigned to the new version is the time when the commit occurs. Each commit is guaranteed to occur at a unique time.

Transactions are identified as queries or updaters. Queries are guaranteed to execute only reads, while updaters may also execute writes. When a transaction starts, it is assigned an access class and a read point equal to the time when it began.

Queries are executed using the MVTO algorithm. The query's read point is associated with each of the read operations executed by the query. Each read of a data item is mapped to the most recently committed version of that data item whose timestamp is earlier than the query's read point. Queries can read any data item whose access class is dominated by the query's access class.

Updaters consist of read and write operations. An updater can read any data item whose access class is dominated by the updater's access class, but it can write only data items whose access class equals its access class. Strict 2PL is used for read and write operations on data items at the updater's access class. These reads and writes are mapped to the latest version of a data item, which may be an uncommitted version created by the updater that is executing the operation. MVTO is used for reads on strictly dominated data items, and an updater's read point is associated with each of these reads. These reads are mapped to the most recently committed version of a data item whose timestamp is earlier than the updater's read point.

Note that a read being executed under MVTO cannot be invalidated by a write from another transaction, because the timestamp associated with the new version of a data item is not assigned until the updater that created it successfully commits. This means that uncommitted versions existing when the read starts will have to receive timestamps greater than the read's read point, and any new updater will have to start after the read's read point.

8.2 Trusted Oracle Concurrency Control and Multilevel Correctness

The Trusted Oracle concurrency control algorithm supports the multilevel correctness properties of single-level serializability and single-level read consistency. The algorithm also mostly satisfies the property of recency. There is at least one way in which progress can be violated, however.

Single-level serializability is efficiently achieved for each access class by the single-level scheduler at that access class using strict 2PL. Each scheduler can perform this task independently without interacting with schedulers at other access classes. Single-level read consistency is enforced for strictly dominated access classes with MVTO. A transaction does not have to worry about the values it reads from strictly dominated access classes becoming invalidated or inconsistent because no transaction from a strictly dominated access class can write a data item version earlier than the original transaction's read point.

A transaction always reads the most recent committed version of a data item. For data items at its access class, the version read is the one that exists when the data item is locked. For data items at strictly dominated access classes, the version read is the last committed version before the transaction's read point. In addition, selecting these versions to be read mostly satisfies recency because they represent the last committed versions that exist before the transaction starts. Numerous intermediate versions could be skipped with this approach. Various modifications may allow slightly more recent versions to be selected, but it is not clear that much would be gained.

While progress is generally supported by Trusted Oracle, there is at least one unusual execution pattern that would cause it to be violated. This situation is depicted in Figure 3. The problem is caused by the order in which two CONFIDENTIAL transactions T_2 and T_3 write versions of y that depend on versions of the UNCLASSIFIED data item z. T_3 begins first and establishes a read line that causes it to read the first version of z. UNCLASSIFIED transaction T_1 then creates a new version of z. T_2 now starts and its read line causes it to read the new version of z. T_2 commits before T_3 and writes a new version of y. T_3 now locks y and when it commits writes a new version of y based on the first version of z. Progress is violated because the version of y written by T_2 depends on a newer version of z than the value of y written by T_3.

This problem occurs because there is a write by T_1 between the two reads. Normally, with locking, T_3 would have taken out a read lock on z which would have prevented T_1 from being able to write. Because this lock cannot be set in a multilevel database, T_1 cannot be blocked. Various modifications could be used to avoid this problem.

Trusted Oracle provides a weaker correctness criterion than multilevel correctness. While it does not guarantee the property of progress, it does appear to provide greater concurrency than algorithms for multilevel correctness. The correctness criteria supported by Trusted Oracle, single-level serializability and single-level read consistency, can be argued to be appropriate for a wide variety of applications.

9 Summary

Almost all database systems use serializability as the definition of correctness for concurrency control. Partially, this occurs because it is the best that can be accomplished without additional semantic knowledge about the database, its operations, or its integrity constraints. Other reasons for using serializability are that it is easy to understand, thoroughly studied, and widely known, and efficient algorithms exist for its implementation.

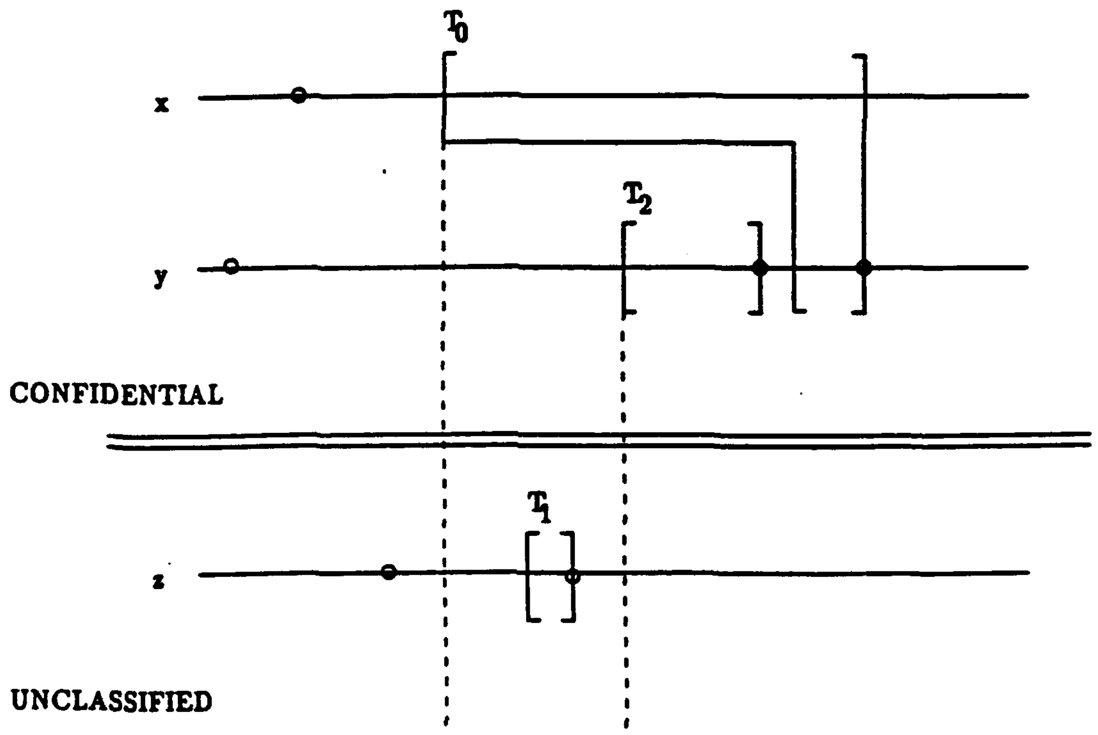


Figure 3: Counterexample for progress in Trusted Oracle

It has been shown that concurrency control interacts with security requirements in a high assurance multilevel database system where each subject is single level. Several secure serializable concurrency control algorithms have been developed for this type of database system, but each exhibits serious drawbacks. When they are based on delaying the execution of a transaction until it does not conflict with strictly dominated transactions, algorithms suffer from performance and denial of service problems. When they are based on forcing a transaction to read old versions of data items so the transaction cannot conflict with strictly dominated transactions, algorithms suffer from low-quality results. These problems for serializable algorithms are a natural consequence of enforcing mandatory security and using single-level subjects, and not just a deficiency of the algorithms that have been developed.

In addition to its effects on serializability, the selected security model also prevents invariant integrity constraints from being enforced across access classes. The semantic knowledge that these invariants cannot exist can be used to develop non-serializable concurrency control algorithms. Because the integrity constraints for the database are guaranteed to not include invariants, the new semantic knowledge does not have to be explicitly specified to or searched for by the database system. The serializable algorithms incur performance or quality problems in an effort to preserve nonexistent invariants. We believe that nonserializable algorithms will be able to perform better by exploiting the new knowledge.

A new nonserializable correctness criteria called multilevel correctness is proposed for concurrency control in multilevel databases. It provides the properties of single-level serializability, single-level read consistency, and progress. The benefits of adding the property of recency are also discussed. Multilevel correctness can be used as the basis for new secure concurrency control algorithms.

Multilevel correctness is then used to examine the nonserializable concurrency control algorithm used by Trusted Oracle. This is the only nonserializable algorithm known for the multilevel database models being studied, and it appears to have good performance characteristics compared to the secure serializable algorithms. We conclude that this concurrency control algorithm observes single-level serializability and single-level read consistency, and performs well with respect to recency, but that there are situations where progress is not enforced.

The Trusted Oracle concurrency control algorithm demonstrates the potential of nonserializable concurrency control algorithms in multilevel database systems. Except in some unusual situations, it implements the properties of multilevel correctness, and it is easy to implement and understand. We believe it is possible to modify the algorithm so that it will also support progress. Further study is needed to develop multilevel correct algorithms, to compare the behavior and performance of secure nonserializable and serializable concurrency control algorithms, and to understand the properties of secure nonserializable concurrency control algorithms.

References

[1] P. A. Bernstein, V. Hadzilacos, and N. Goodman. *Concurrency Control and Recovery in Database Systems.* Addison-Wesley, 1987.

[2] Dorothy E. Denning, Teresa F. Lunt, Roger R. Schell, William R. Shockley, and Mark Heckman. The SeaView Security Model. In *Proceedings of the 1988 IEEE Symposium on Security and Privacy*, pages 218–233, Oakland, CA, April 1988.

[3] Alan R. Downing, Ira B. Greenberg, and Teresa F. Lunt. Issues in Distributed Database Security. In *Proceedings of the Fifth Annual Computer Security Applications Conference*, pages 196–203, Tucson, AZ, December 1989.

[4] Ira B. Greenberg. Distributed Database Security. Final report, SRI International, Menlo Park, CA, April 1991. For Contract No. MDA904-90-C-7708.

[5] T. F. Keefe and W. T. Tsai. Multiversion Concurrency Control for Multilevel Secure Database Systems. In *Proceedings of the 1990 Symposium on Security and Privacy*, pages 369–383, Oakland, CA, May 1990.

[6] H. T. Kung and C. H. Papadimitriou. An Optimality Theory of Concurrency Control for Databases. In *Proceedings of the 1979 ACM-SIGMOD International Conference on Management of Data*, pages 116–126, Boston MA, May 1979.

[7] H. T. Kung and J. T. Robinson. On Optimistic Methods for Concurrency Control. *ACM Transactions on Database Systems*, 6(2):213–226, June 1981.

[8] T. F. Lunt, R. R. Schell, W. R. Shockley, M. Heckman, and D. Warren. A Near-Term Design for the SeaView Multilevel Database System. In *Proceedings of the 1988 IEEE Symposium on Security and Privacy*, April 1988.

[9] Teresa F. Lunt. Multilevel Database Systems: Meeting Class A1. In *Proceedings of the Second IFIP Database Security Workshop*, October 1988.

[10] William T. Maimone and Ira B. Greenberg. Single-Level Multiversion Schedulers for Multilevel Secure Database Systems. In *Proceedings of the Sixth Annual Computer Security Applications Conference*, pages 137–147, Tucson, AZ, December 1990.

[11] William R. Shockley and Roger R. Schell TCB Subsets for Incremental Evaluation. In *Proceedings of the Second AIAA Conference on Computer Security*, December 1987.

Using TNI Concepts for the Near Term Use of High Assurance Database Management Systems

Cynthia E. Irvine, Roger R. Schell, and Michael F. Thompson
Gemini Computers, Inc.
P.O. Box 222417, Carmel, CA 93922

Abstract

High assurance multilevel database management systems offer attractive solutions to a variety of operational data management problems. There is a need to field viable database management systems soon. The monolithic evaluations dictated by the Trusted Database Interpretation will prohibit their emergence through that avenue in the immediate future. Class A1 trusted network systems are currently feasible which can achieve the realization of high assurance database solutions.

The Trusted Network Interpretation allows various components which are separately evaluated to be used as the building blocks of a complete system. In this paper a hypothetical database component will be discussed in the context of systems at the highest levels of assurance. The features of this component will be described and its advantages presented. Among the latter are the elimination from the DBMS of a need for trusted path support, identification and authentication, and labeling of human readable output.

An embedded database management system being developed at Gemini will be used as an example of a possible database component.

1 Introduction

The Trusted Network Interpretation (TNI) of the Trusted Computer System Evaluation Criteria (TCSEC) [1] provides interpretations of the TCSEC [2] appropriate for evaluating a network system as a single system with a single Network Trusted Computing Base (NTCB) that is physically and logically partitioned among components of the network. The Trusted Database Management System Interpretation (TDI) of the TCSEC [3] provides interpretations appropriate for evaluating database management systems against the requirements of the TCSEC. Neither the TDI nor the TNI provide explicit guidance for the evaluation of network systems containing databases. While not specifically excluding such systems neither interpretation provides explicit guidance for evaluating database systems that are partitioned into components of a network. In particular, the TNI does not address the evaluation of components that contain TCB subsets and the use of a subset evaluation for components. This paper outlines an approach toward evaluation of such systems that is consistent with the techniques and requirments of both interpretations.

As an example, consider a network component to which has been allocated a discretionary access control policy. A requirement of the TNI is that the network component meet the TCSEC DAC requirement. Assuming the network component is a database management system, the TDI provides an interpretation useful for evaluating compliance with the DAC requirement for database management systems. Once established, (and the other requirements of Appendix A of TNI are met) the component may be treated as a "D-Component" of the TNI, and composed, in accordance with Appendix A of TNI, with other network components such that an overall network security policy is enforced.

This paper describes the network security architecture of the "Embedded Secure DBMS" (ESDBMS), a network component that is a composition of two distinct components: 1) a "DBMS component" that includes the ORACLE Relational DBMS; and 2) an M-component, the Gemini Trusted Network Processor (GTNP),that incorporates the GEmini Multiprocessor Secure Operating System (GEMSOS) [4] Mandatory Security Kernel [5]. The ESDBMS component, which is designed to meet Class A1 requirements for MDA-Components as defined in the TNI, provides multilevel secure database management. In addition to a Class A1 M-component enforcing a Mandatory Access Control (MAC) policy, the ESDBMS includes enforcement of a Discretionary Access Control (DAC) policy and auditing. Other components provide direct user connections, i.e., user identification and authentication. These other network components may be mainframe host computers that each run at a single security level.

Section 2 of this paper presents a statement of the problem that the ESDBMS is intended to solve and identifies product features of the ESDBMS and describes its intended use in terms of candidate system applications. Concepts found in the TDI and TNI which provide the basis for the evaluatability of the system encompassing the ESDBMS are presented in Section 3. Section 4 presents a formulation of the ESDBMS architecture within the context of the TNI, and describes how the TNI is a useful tool for solving this class of problems. Section 4.1 reviews the TNI support for the composition of separately rated components into an evaluatable network system, and section 4.2 introduces a network architecture that includes distinct network components, including a DBMS component. Section 4.3 then proposes an implementation of the DBMS component as a virtual machine that executes on the GTNP. The architectural properties of the DBMS component necessary to support composition with the GTNP M-Component are presented in section 4.4, followed by a discussion of how the TDI may be used to evaluate the DBMS component against TCSEC requirements in section 4.5. Section 4.6 then presents the result of composing the DBMS component with the GTNP, namely the ESDBMS component, in terms of its network security architecture. Section 4.7 identifies some future enhancements to the ESDBMS. Finally, section 5 presents the conclusions.

2. ESDBMS Features and Candidate Applications

Command, Control, and Communication (C^3) computer systems manage large volumes of diverse operational data in support of human, automated, or semi-automated decision making, while serving an operational need for rapid access to timely information. To date, the automation of many C^3 functions has been hampered by several factors: the need to protect information at a variety of classification levels from unauthorized modification or disclosure; the need to ensure that critical transactions based on "up to date" data are performed in a time-critical manner; and the need to develop customized systems meeting the requirements of specific installations.

The ESDBMS is designed to serve as a DBMS "backend" database engine providing rapid access to shared information at various classifications and time-critical transaction processing while enforcing a security policy for the access of users to information with a high degree of assurance. Intended to serve as a Class A1 MDA-Component as defined in Appendix A of the TNI, the ESDBMS is a synergistic combination of the GEMSOS Mandatory Security Kernel, packaged within the GTNP, and the ORACLE Relational DBMS. When combined with suitable Class C2+ host front ends, the ESDBMS results in a complete Class A1 system. The ORACLE RDBMS [6] was particularly suited to the architectural approach of the ESDBMS because it is read lock-free and allows arbitrary transaction suspension. Other database approaches [7] were possible, but only ORACLE provided a range of capabilities required by potential users. By utilizing a general-purpose commercial DBMS already supporting utilities, applications and accepted standards (e.g., ANSI SQL), the ESDBMS has the potential to provide a highly versatile and functional base for C^3 operations with high assurance security.

The ESDBMS is intended to be available on the compatible family of commercial off-the-shelf GTNP products, which have performance capacities ranging from 0.5 MIPS to 25 MIPS. In addition, the GTNP product family includes a variety of storage and interface devices, including Ethernet, HDLC, IBM High Speed Channels, and up to sixty-four RS-232 serial ports.

A major problem that a general DBMS tailored for C^3 applications must solve is that of enforcing, with a high degree of assurance, mandatory and discretionary security policies for the observation and modification of data. The ESDBMS is intended to support system operational concepts such as those requiring operators with different security clearances and "need-to-know" permissions. The tools for the design of data and its manipulation offered by a commercial DBMS should also be tailored to recognize the potential existence of data of different sensitivities in order to realize the maximum cost leverage possible in the face of escalating application software development and maintenance costs.

The class of application the ESDBMS is intended to serve can be characterized as being required to meet a rigid time-critical schedule (e.g., guaranteed end-to-end response time) for certain high-priority predefined transactions that are executed in response to events external to the system. The end-to-end processing time must be predictably met under "worst case" conditions of system load. In contrast with the ESDBMS design, DBMSs hosted on multi-user, general-purpose operating systems generally cannot offer time-critical service because the underlying operating system does not offer the ability to dedicate resources to high-priority transactions as they execute. Thus without products such as the ESDBMS, the range of C^3 applications that can be contemplated is limited to those where the operational benefit justifies the development of application-specific time-critical data management routines as part of the procurement. In such systems, conventional DBMS packages are typically only usable to support non-critical, transaction processing [6, 8].

The architectural approach of the ESDBMS requires transaction suspension and concurrent execution of transactions at differing access classes. The read and rollback strategy of the Trusted ORACLE RDBMS allows concurrency in a multilevel environment. Each process obtains its own consistent copy, or snapshot, of whatever portion of the database it is currently accessing. To read a particular tuple, "undo" data is used to "restore" the database to the state it was in when the process started the transaction. Thus, for a given process, the database appears unchanged as of the start of the transaction. Within a particular ongoing transaction, writing to the database affects only the process local snapshot.

ORACLE provides a version number for each transaction which identifies the state of the tuple at the time the transaction was initiated. This allows the read and rollback strategy to be effective and allows each transaction to complete [9].

Potential applications of the ESDBMS include command and control information systems such as that designed to serve SACLANT [10]. The SACLANT system requires a multilevel secure database containing, among other things, platform tracking information. In this system priorities are associated with each query, and high priority transactions have rigid response time requirements that are independent of the system's load of lessor priority queries.

C^3 systems often include a requirement for the enforcement of a mandatory integrity policy [11], allowing mission-critical data to be modified only by operators and application programs that are trusted to make such modifications properly. As an example of the utility of a mandatory integrity policy for C^3 applications, consider thé usual objection to the notion of supporting "user-written" applications accessing C^3 data, that there is no assurance that such applications, if erroneous and being executed by a subject at a high security level, will not destroy or degrade the data upon which critical system transactions depend. If a mandatory integrity policy is enforceable, "damage control" boundaries can be erected around critical data such that no user program, however erroneous it may be, will be allowed to modify the data. Similar protections against damage can be offered for the programs delivered with the system.

Another example of the need for mandatory integrity policy enforcement can be found in systems with multiple modes of operation. An example is that designed for SACLANT that are intended for use in training exercises as well as for operational purposes. Use of mandatory integrity would allow operational data to be observed but not modified as a result of exercise and training activity.

A further class of potential applications of the ESDBMS are general data services environment in which mainframe computers execute at different security levels. In such an environment, the ESDBMS would allow sharing of information between the single level mainframes; giving users a "single system view" of information at differing security levels. An example of such a system is that being installed as part of the Headquarters System Replacement Program (HSRP) [12]. If these capabilities are combined with an appropriate interface such as Multilevel SQL (MSQL), which is being developed for the SeaView implementation prototype [13], a high utility system will result.

3 Conceptual Foundations for the ESDBMS

The problem of achieving high assurance multilevel database capabilities has been recognized since the "Summer Study" that occured in 1982 [14]. It appeared that the design multilevel database systems would result in architectures in which significant portions of the database were trusted, with the concomitant result that such systems would unlikely to be evaluatable at the highest levels of assurance. Major advances in high assurance database technology resulted from the SeaView Class A1 Relational DBMS project [15, 16]. Generalization of the standard data model to a multilevel environment combined with an architecture which allowed the database to execute outside the perimeter of a mandatory security kernel while providing adequate functionality demonstrated the promise of evaluatable high assurance database managment systems. The evaluatability of such systems is grounded in the notions of TCB subsets and balanced assurance. This section will provide an overview of these concepts.

3.1 TCB Subsets

The concept of TCB subsets was formally defined as a method for decomposing a TCB into components which could be separately evaluated and then combined into a complete TCB [17]. An abbreviated definition of TCB subsets is provided in the TDI. It is possible to establish that the ESDBMS architecture satisfies the conditions for TCB subsetting by meeting the following criteria for a coherent subset architecture. These criteria are outlined here and will be described more fully in later sections within this paper.

- identification of candidate TCB subsets
- "allocation" of system policy to the candidate TCB subsets
- explicit description of the TCB subset structure, or architecture
- substantiation of the assertion that the TCB subsets occupy distinct "subset domains"

The historical context for the ESDBMS is found in classical Hinke-Schaefer architectures [18] for multilevel database systems: DBMS executes as an untrusted application on top of a high assurance TCB. In the ESDBMS, a high assurance mandatory TCB supports the execution of a trusted DBMS.

Although it does not comprise a complete TCB, the ESDBMS can be viewed as containing two candidate TCB subsets: a mandatory subset, and an audit and DBMS DAC subset. The policy allocation to the subsets will be such that the mandatory technical policy will be enforced by the mandatory component, while the audit and database discretionary policies are enforced by the non-mandatory TCB subset. The architecture is such that the subset enforcing the mandatory access control policy will be more privileged that the non-mandatory TCB subset. Finally it will be demonstrated that the mandatory TCB subset creates the notion of distinct domains for each less privileged subset and that the domains are hierarchical such that each subset is non-bypassable, maintains disjoint objects, and is self protecting.

3.2 Balanced Assurance for Near term Solutions

The notion of *balanced assurance* was introduced with SeaView [19, 20] and provides a methodology for achieving a high level (Class A1) of assurance for a system as a whole by applying high assurance techniques to the part of the system enforcing the mandatory access control policy while requiring assurance measures equivalent to C2 for those parts of the TCB enforcing the non-mandatory access control policy. In contrast, *uniform assurance*, requires the assurance measures of the target evaluation class to be applied to each TCB subset (or, in a collapsed TCB, to a monolithic TCB).

The application of balanced assurance is based on the strict allocation of policy to TCB subsets for which it can be demonstrated that the TCB subsets occupy distinct protection domains. Each TCB subset will enforce a clearly expressed policy on its subjects. In addition, since they are in distinct protection domains, hierarchically ordered (i.e., more privileged) TCB subsets are absolutely tamperproof with respect to less privileged TCB subsets and hierarchically unrelated TCB subsets are absolutely unrelated to each other. (The latter can be characterized as similar to the isolation provided by a Virtual Machine Monitor for the Virtual Machines it creates.) By meeting these subsetting requirements, it is possible to state that the non-mandatory TCB is constrained by the subset enforcing mandatory security policy and is unable to compromise mandatory security.

The TCB subset enforcing mandatory access control policy and any TCB sets it relies on must be subjected to a uniform level of assurance requirements, which will be level of the target evaluation class. Thus, any TCB subsets enforcing the system', mandatory access control policy must satisfy the full gamut of assurance requirements of the target evaluation class. It can be argued that to apply high assurance techniques to the non-mandatory TCB subsets in no way increases the assurance that the mandatory access control policy is correctly enforced. An explicit characteristic of a manadatory access control policy is that the nature of the information it is intended to protect is such that its disclosure will result in a compromise to security for which the penalties are severe. In contrast, the penalties meeted out for infractions of discretionary access control policies are less harsh. Therefore, one can posit that the policy which must be enforced with the highest degree of assurance must be the mandatory policy, while the discretionary policy enforcement mechanism may be subjected to a less stringent group of assurance requirements. Hence, TCB subsets enforcing any non-mandatory access control policies have a benefit of a relaxation of the requirements for applicable assurance techniques. For example, the Class C2+ assurance requirements of the TNI require non-mandatory TCB subsets to meet the target evaluation class requirements for functionality, but assurance requirements for Class C2.

The TDI does not define the basis for evaluations to be conducted using balanced assurance: database systems must be evaluated monolithically. Two important factors contribute toward the infeasibility of high assurance evaluations of data base management systems at this time. The first is the state of the art for verification for which certain concerns remain.

The second relates to serious fundamental issues of computability of the correspondance to a model. A feature of sophisticated database systems is that they support the enforcement of content-dependent DAC policies, for example, through view-based access control. The content dependance of such policies make them, by their very nature, dynamic. One could consider the situation similar to one in which one has an arbitrary program and an arbitrary specification. In the case of completely general view-based access control, there is an arbitrary policy and an arbitrary mechanism. It is not clear that it is possible to provide correspondence to a formal model as evaluation evidence *a priori*. There are substantial parallels between the problem of database DAC policies and that examined by Harrison, Ruzzo and Ullman [21] regarding the undecidability of the safety of a system where discretionary access controls are used to determine access to objects. They proved that the safety problem could be reduced to the halting problem. Database DAC policies face similar problems of non-computability.

It is unlikely that major commercial vendors will be willing to commit corporate resources to a high assurance evaluation at a time when the outcome of the evaluation is in doubt due to current status of formal methods as well as problems on non-computability intrinsic to database systems. This means that although they will be evaluated at lower levels of assurance, it is unlikely that a commercially viable DBMS, such as the ORACLE RDBMS, will be evaluated at Class B3 or A1 under the TDI. The notion of balanced assurance is encompassed (although not explicitly by name) in the TNI, which presents an attractive alternative for achieving a high assurance evalution.

4 The TNI is a Vehicle for a Solution

This section describes how the TNI can be used to evaluate the ESDBMS.

The TNI provides interpretations of the TCSEC appropriate for evaluating a network system as a single system with a single NTCB that is physically and logically partitioned among components of the network. In Appendix A of the TNI, this view is extended to one in which a network system is partitioned into components; each component is rated to determine its security relevant characteristics; and then the composition of the components is evaluated to arrive at an overall rating class for the network. As noted in section A.1 of the TNI:

This approach aids in the assigning of an overall network evaluation class in two ways: 1) it allows for the evaluation of components which in and of themselves do not support all policies required by the TCSEC (which will then contribute to the overall evaluation of any network which uses the *evaluated component), and 2) it allows for the reuse of the evaluated component in different networks without the need for a reevaluation of the component.*

This TNI approach is one in which an individual component's NTCB "partition" (viz., that component's hardware, software and firmware responsible for enforcing the particular subset of the overall network security policy allocated to that component) is evaluated against TCSEC requirements applicable to those polices enforced by that component.

Thus, once a set of components are evaluated against their applicable TCSEC requirements, the TNI provides the tools necessary to compose the components into an evaluatable system that enforces a complete network security policy. The TNI is not prescriptive with respect to the manner in which components are evaluated against TCSEC requirements. We assert that the TDI provides an appropriate interpretation for evaluating a DBMS component against the TCSEC requirements applicable to that component as defined by the TNI for DA-Components. Use of the TDI for evaluation of the DBMS component is described in section 4.5

4.1 TNI Composition Rules

Appendix A of the TNI provides guidance for evaluation of the individual components of a trusted network and defines a set of rules for the composition of evaluated components to form an evaluatable (sub)system and the method for assigning a rating to a (sub)system conforming to the rules. For example, section A.2.9.1 of that Appendix describes the composition of a Class A1 M-Component (a component that enforces a mandatory access control policy), and a Class C2+ D-Component (a component that enforces a discretionary access control policy having the functionality required of Class B3 systems and the assurance required for Class C2 systems). This composition results in a Class A1 MD-Component (a component that enforces both mandatory and discretionary access control polices.)

4.2 Network Architecture Overview

This section presents a network architecture that includes DBMS components. Figure 1 illustrates the architecture in terms of the network components and their allocated functions and policies. Note that Figure 1 illustrates the logical architecture, not necessarily the physical architecture. In it a set of Class C2+ DBMS components having different security levels are connected through a GTNP designed to meet the Class A1 requirements for a M-Component of the TNI. The GTNP also provides connections for the DBMS components to a set of Class C2+ host computer systems having directly connected users.

In Figure 1, each Class C2+ component has a single security level, and is connected to the GTNP through interface connections at that single security level. The DBMS components enforce DAC and audit policies over the databases based on IDs associated with specific communication channels, and the DBMS components perform audit. The host components authenticate users; perform audit; and enforce a DAC policy that includes discretionary authorizations of subjects to attach to the channels that connect to the DBMS components via the GTNP M-Component.

The specific network security architecture of the DBMS is further described in section 4.4, however first a specific class of network components, "virtual machines" are discussed and the DBMS component is shown to be such a component.

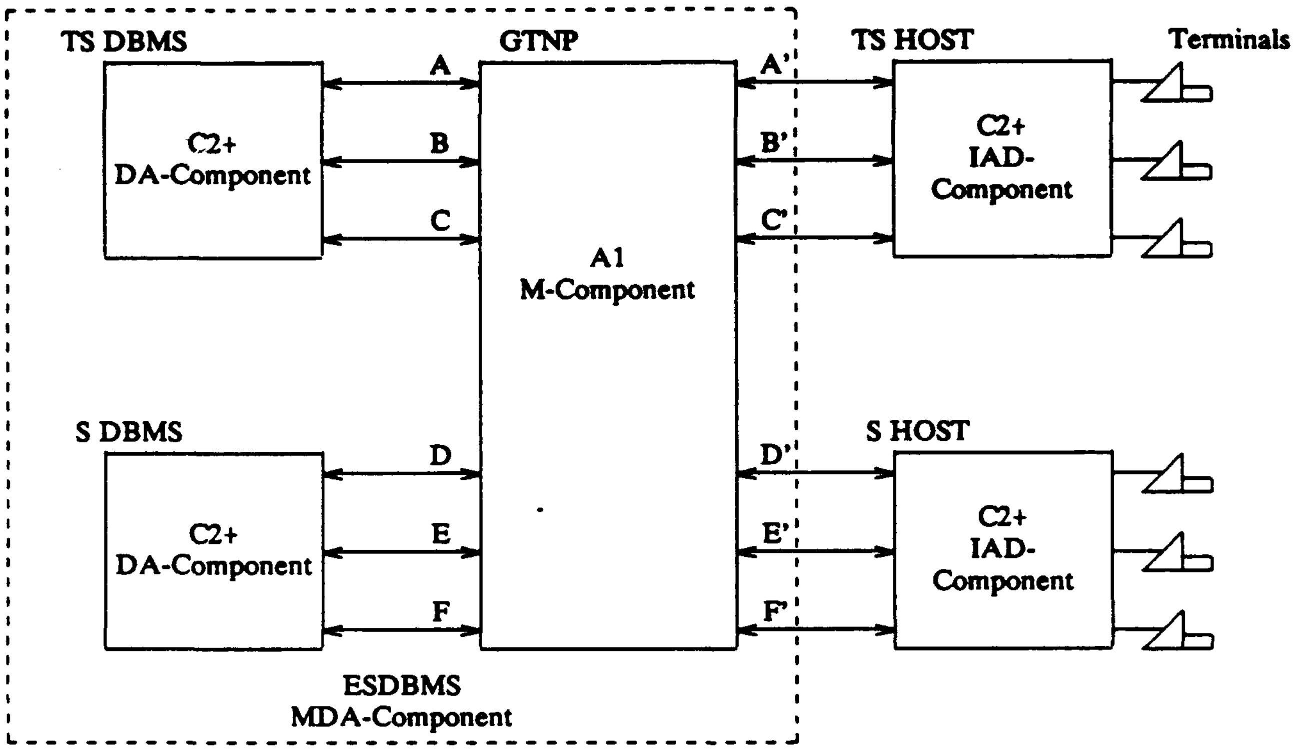


Figure 1

Composition of a Gemini Trusted Network processor M-Component with Class C2+ DA-Components in a MDA-Component designed to meet Class A1 requirements

4.3 Virtual Machines as Network Components

As described in the Introduction to the TNI, a virtual machine is an example of a network component. In the architecture illustrated in Figure 1, the DBMS components execute as virtual machines created by the Class A1 M-Component. To include explicit support for multi-state virtual machines, the GTNP provides mechanisms allowing a comprehensive ring integrity policy to be implemented [22,23]. Each virtual machine executes at a single, distinct access class and sharing between virtual machines is mediated and controlled by the M-component such that the state of one machine does not affect the GTNP's access mediation with respect to other machines. Within each of these virtual machines, the GTNP provides the support required to create hierarchically ordered privilege domains such that more privileged domains are absolutely tamperproof with respect to those that are less privileged. It is a property of the GTNP's network security architecture and design (as described in the TNI Introduction) that virtual machines constructed with this ring mechanism are intended to be evaluatable as distinct network components.

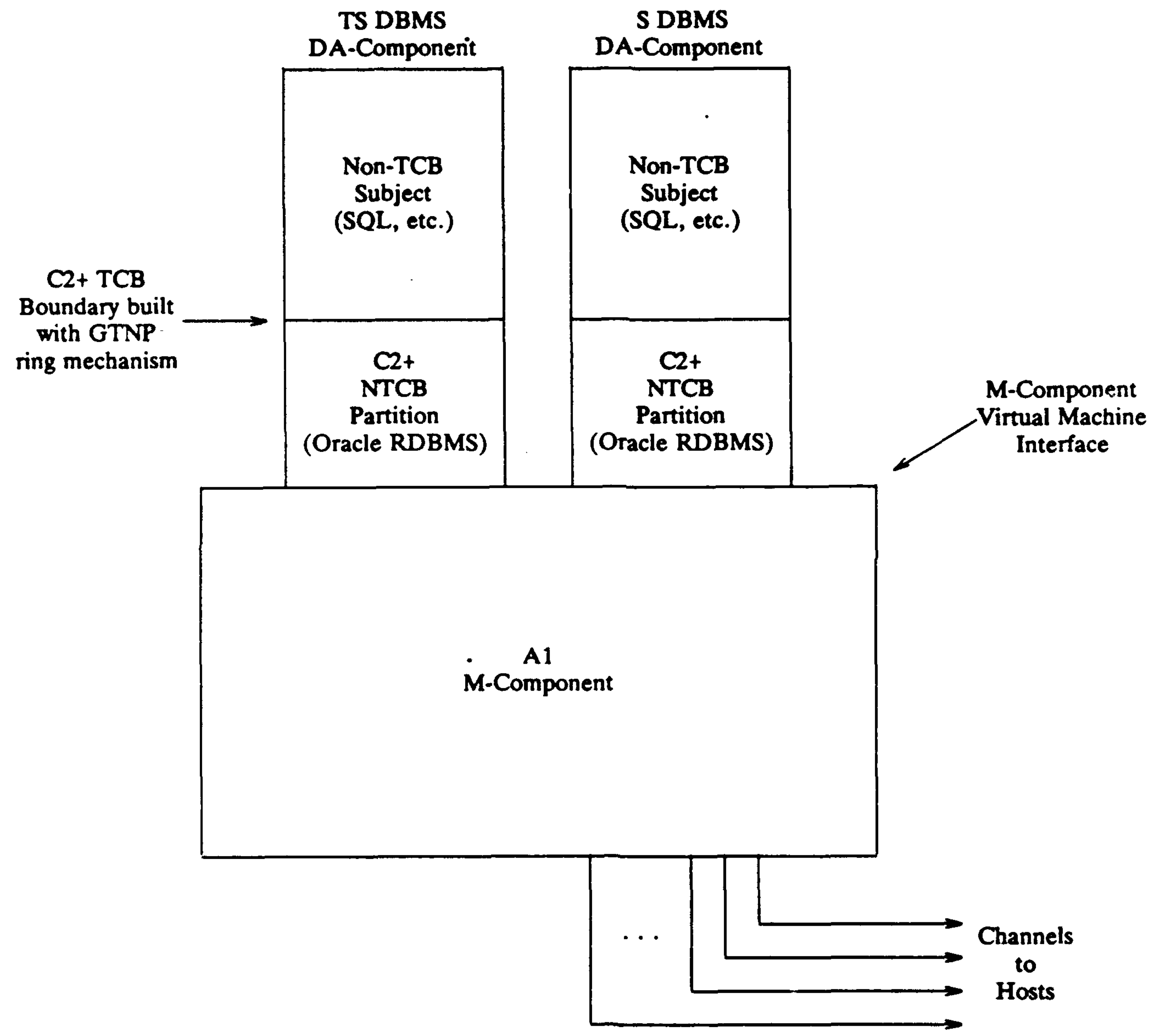


Figure 2

The GTNP supports virtual machines as distinct network components.

Each DBMS component virtual machine is implemented as a process on the GTNP. Outside the GEMSOS Mandatory Security Kernel, each process has multiple dominance domains, where the domain containing the DBMS component's NTCB partition is more privileged than that containing the DBMS component's non-TCB subjects.

The GTNP provides a single level interface to network components implemented as virtual machines on the GTNP. In Figure 1 these single level interfaces between the M-Component and the DBMS components are represented by communication channels labeled "A" through "F". The virtual machine interface presented by the GTNP M-Component is composed of primitive operations (i.e., non-privileged machine instructions and GEMSOS Kernel functions) that may be used to construct an inter-component protocol for use by components when communicating via the GTNP. This protocol will utilize shared segments and synchronization primitives (viz., eventcounts and sequencers [24].)

Figure 2 illustrates the DBMS components executing as virtual machines on the GTNP M-Component. Composing the GTNP M-Component and the DBMS DA-Components results in the ESDBMS MDA-Component. To perform this composition in accordance with the TNI, the DBMS component must first be shown to be consistent with a specific network security architecture. This is discussed in the following section.

4.4 Network Security Architecture: DBMS Composition Requirements

This section identifies necessary architectural features of the DBMS DA-Component such that it may be composed with the GTNP M-component in accordance with the system architecture considerations described in Appendix A of the TNI.

The DBMS DA-Component network security architecture reflects a correct use of the M-Component's ring mechanism to implement dominance domains [22], thus providing the DA-component with a domain for its execution and protecting the component's NTCB partition. The DBMS DA-Component NTCB partition uses M-Component resources to present an abstraction of subjects and objects at its interface. Examination of the resources presented by the M-component interfaces and their utilization by the DBMS-component demonstrates that the subjects and objects exported by the latter are based exclusively on the resources and functional support provided by the former. The DA-component is designed to provide the association of an identifier, used for DAC enforcement, with the subjects attached to the various communication channels connecting the DBMS to the host computers. Finally, its audit capabilities include audit records based on error messages returned to it by the M-Component (e.g., due to attempts to violate the MAC policy.)

The characteristics of the DBMS DA-component described above are sufficient to allow that component to meet the composition requirements of the TNI as described in its Appendix A. These are in addition to those required to meet the DAC and Audit requirements of the TCSEC.

4.5 Evaluatability of the DBMS as a TDI-TCB Subset

It is necessary to establish that the DBMS, in this case the ORACLE RDBMS, which provides the network DA-component, is designed to meet both the DAC and Audit requirements of the TCSEC. The TDI explicitely introduced the notion of TCB subsets so that database evaluations could be conducted incrementally through an evaluation by parts. Thus a DBMS evaluation could be achieved through two separate evaluations: that of an underlying operating system which satisfies the criteria for a TCB subset enforcing mandatory security policy, and then of the DBMS as a less primitive subset utilizing the subjects, objects, and functional interface presented by the more primitive subset. (Readers should be careful to note the use of the word "primitive" in the TDI in contrast to earlier discussions [17] of TCB subsets which used the more standard notion of "privilege.")

The ORACLE RDBMS [6] is designed to meet the criteria for a TCB subset and is evaluatable as such under the TDI. The principle characteristics of the ORACLE RDBMS that make it especially appropriate for the ESDBMS are its "distributed architecture," which permits TCB subsetting; the absence of "read locks;" its capability for arbitrary transaction suspension; its Class C2+ DAC meeting the requirements of the TCSEC; its well-defined programmably isolated interface; and its C^3 capabilities. This evaluation will be sufficient to demonstrate that the DBMS meets the DAC and Audit requirements of the TCSEC.

4.6 Network Security Architecture of the ESDBMS

Once the DBMS component's network security architecture is shown to have the architectural characteristics identified in section 4.4, the GTNP and a set of DBMS components may be composed into a Class A1 MDA-Component. This section describes the network security architecture of the ESDBMS, viz., its security functions and interfaces to other network components.

The security functions of the ESDBMS are the enforcement of mandatory and discretionary access control policies and the performance of audit.

All mandatory access control policy within the ESDBMS is provided by the GTNP, thus its MAC policy is described completely by the GTNP. The mandatory security kernel of the Gemini Trusted Network Processor supports the enforcement of policies regarding the unauthorized disclosure of information (secrecy) and the unauthorized contamination of information (integrity). The mandatory security mechanism is based on controlling the access of subjects with a given sensitivity level to protected information having a specific access class label. The GEMSOS access labels may be used to create sixteen hierarchical secrecy protection levels and sixteen hierarchical integrity protection levels. Two category divisions are available to augment the two types of protection levels: one division of 64 secrecy categories and a second group of 32 integrity categories. These levels and categories form a partially ordered set. Label-based policies that may be represented by a sublattice of the complete lattice may be implemented.

The DBMS enforces discretionary policies such as access to relations, authorization to read, update, or modify database entities, etc. The DBMS enforces constraints on the data. These include integrity control mechanisms for referential integrity, recovery, concurrency support, and transaction control.

Additionally, the ESDBMS DA-component performs audit. These audit operations can be divided into two categories: those stored in a system-high audit trail which record operations which are external to the database, such as requests to create a database; and those stored by the ORACLE RDBMS in an instance-level audit trail which will record operations which are internal to the database. The ORACLE audit trail can provide a fine grained record of access to tables or views and the use of different types of SQL operations.

The principle component interface presented by the ESDBMS is a set of single level channels, each of which is dedicated to a specific subject ID as determined by a security administrator to represent users, groups of users, or both. These single level channels are intended to connect to single level host computers.

The IDs associated with each single level channel are administratively mapped to whatever authorizations are used to control access to the channels on the connected host computers (the connections labeled A' through F' in Figure 1.)

5 Potential Future ESDBMS Enhancements

The ESDBMS as discussed to this point is that which is being developed as part of a Phase II SBIR contract through Rome Air Development Center. It has been dubbed the "primitive option" because it provides only the functionality necessary to support basic multilevel secure embedded database features. A number of more advanced features have been identified.

5.1 Multilevel Device Interface

As described above, the ESDBMS does not support multilevel channels as defined in the TNI. Thus each host with directly connected users must interface to the ESDBMS through single level channels. This limitation is inherent when the connected hosts are single level. However, if one introduced multilevel hosts to which users could negotiate sessions of different security levels, the single level channels could become limiting. The ESDBMS could be extended to support multilevel channels for use in communicating with multilevel hosts. The GTNP's ring mechanism can be used to introduce additional virtual machines that implement multilevel channels. In this specific extension, a ring would be defined that is a more privileged domain than that to which the C2+ DBMS NTCB partition is allocated. This ring would contain the implementation of a multilevel subject that presents the multilevel channel abstraction, as suggested by the Rational of section 4.1.1.3.2.1, "Exportation to Multilevel Devices" of the TNI.

A multilevel service can be tailored to meet the specific needs of the ESDBMS. This will be a "channel receive" process able to allow connections to be made to the ESDBMS from remote systems. By recognizing the access class of the connecting system, the channel receive process must be able to activate a single level subject within the ESDBMS which will process the requested transaction. It is intended that the channel receive process establish the priority of the requested transaction such that high priority transactions are guaranteed a response time based on "worst case" analysis.

As an adjunct to this enhancement, the system could be adapted to provide for dynamic DBMS instance creation, thus yielding a solution to the "Gzillion Problem" [25]. This problem is encountered in systems intended to support a potentially very large number of access classes. To provide for each possible access class statically would consume system resources that might never be used by an actual transaction. The solution is twofold: data structures are designed so that per access class information is created and locatable algorithmically, and a multilevel subject is provided that is able to create single level instances of the data base at any of the possible access classes. The primary features of the Trusted ORACLE RDBMS that would require modification to support dynamic instantiation of the database at new access classes have been identified.

Methods for view expansion will support subjects occurring at previously unencountered access classes such that these subjects are provided with valid views in the allowable range of readable access classes.

5.2 Adaptive Priority Scheduler

Currently, when the process with the highest priority obtains the processor, it executes until it relinquishes the processor voluntarily or as a result of preemption by a higher priority process becoming ready to run. Some environments may require the modification of this paradigm. An adaptive priority scheduler could be included so that within a particular priority level, all transactions are given an equal time share. An adaptive scheduler could permit time slicing at lower priorities while maintaining the preemptive nature of the higher priority transactions.

5.3 High Granularity Discretionary Access Controls

Each interface (viz., single level channel) to the ESDBMS is dedicated to a single ID (either a user ID, group ID, or both) for purposes of discretionary access control. It is required that this ID be administratively mapped to the discretionary authorizations associated with the device interfaces on the single level hosts. A future enhancement to the ESDBMS would be to extend the C2+ DBMS NTCB partition on the ESDBMS to dynamically associate IDs with information read and written from the channels. This enhancement would require that the connected hosts incorporate a similar protocol for associating information with users.

The ESDBMS enforces a discretionary access control policy as provided by the Oracle RDBMS. This DAC is implemented to a Class C2+ level of assurance as defined in the TCSEC in combination with the TNI. A possible extension to the ESDBMS is to introduce an additional virtual machine that enforces a DAC policy on named objects that are used by the DBMS to implement the database itself.

4 Conclusion

For the foreseeable future, it is infeasible to produce a high assurance database management system in which the database "engine" itself is trusted at the higher levels of assurance. In addition to issues of state of the art for verification, there are serious fundamental issues of computibility of the correspondence to a model, even at Class B2. As a consequence, near term evaluations of monolithic trusted database management systems under the auspices of the TDI must be restricted to low assurance systems.

The principle objective of the ESDBMS design is to provide an overall architecture that allows the processing of high-priority, application-specific, preprogrammed DBMS transactions with a guaranteed worst-case end-to-end processing time in a high assurance multilevel secure context. This paper has outlined an approach to achieving an evaluatable high assurance DBMS in the near term by using an architecture based on TNI network components, some of which are DBMS components. This is possible because the TNI provides a set of objective metrics by of separately evaluated components having differing assurance classes can be combined into network architecture components and evaluated as a high-assurance trusted network.

The authors gratefully acknowlege support from the U.S. Air Force, Rome Air Development Center, under contract F30602-90-C0117.

References

1. *Department of Defense Trusted Network Interpretation of the Trusted Computer Evaluation Criteria.* 31 July 1987, NCSC-TG-005 Version-1, National Computer Security Center, Fort Meade, MD 20755-6000.

2. *Department of Defense Trusted Computer System Evaluation Criteria.* DoD 5200.28-STD, December 1985. National Computer Security Center, Fort Meade, MD 20755-6000.

3. *Department of Defense Trusted Database Management System Interpretation of the Trusted Computer System Evaluation Criteria*, April 1991. NCSC-TG-021, National Computer Security Center, Fort Meade, MD 20755-6000.

4. Shockley, W.R., Tao, T., and Thompson, M.F., *An Overview of the GEMSOS Class A1 Technology and Application Experience.* In **Proc. 11th National Computer Security Conference**, October 1988.

5. Thompson, M.F., Schell, R.R., and Levin, T.E., *Introduction to the Gemini Trusted Network Processor.* In **Proc. 13th National Computer Security Conference**. October 1990.

6. Irvine, C.E., Schell, R.R., and Vetter, L.L., *Architecture for an Embedded Secure Data Base Management System*, In Proc. of AIAA/ASIS/IEEE **Sixth Aerospace Computer Security Applicaitons Conference**, December 1990.

7. Lefkovits, H.C., Schell, R.R., Gajnak, G.G., Shockley, W.R. *The Multilevel Secure Entity-Relationship DBMS.* **RADC-TR-88-310**. Rome Air Development Center, Griffiss AFB. January 1989.

8. Irvine, C.E. and Towle, B.S., *Embedded Secure Database Management System*, **RADC-TR-90-439**, December 1990.

9. Maimone, W., and Greenberg, I., *Single Level Multiversion Schedulers for Multilevel Secure Database Systems.* In Proc. AIAA/ASIS/IEEE **Sixth Aerospace Computer Security Applications Conference**. May 1990.

10. Schell, R.R., Thompson, M. F., and Shockley, W.R., *A Network of Trusted Systems,* In **Proc. of AIAA/ASIS/IEEE Third Aerospace Computer Security Conference**, December 1987.

11. Schell, R.R., and Denning, D.E. *Integrity in Trusted Database Systems.* In **Proc. 9th National Computer Security Conference. 30**, 1986.

12. Gambel, D. and Walter, S., *Retrofitting and Developing Applications for a Trusted Computer Base*, In **Proc. 11th National Computer Security Conference**, October 1988.

13. Lunt, T.F., *Security in Database Systems: A Researcher's View*, In **Proc. of the Second German Conference on Computer Security**, 1991.

14. Committee on Multilevel Data Management Security. *Multilevel Data Management Security.* Air Force Studies Board, National Research Council, National Academy Press, 1983. For Official Use Only.

15. Lunt, T.F, Neumann, P.G., Denning, D.E., Schell, R.R., Heckman, M. and Shockley, W.R., *Security Policy and Interpretation for a Class A1 Multilevel Secure Relational Database System.* Computer Science Laboratory, SRI International, SRI-CSL-88-8, August 1986.

16. Denning, D.E., Lunt, T.F , Schell, R.R., Heckman, M., and Shockley, W.R., *Secure Distributed Data Views (SeaView): The SeaView Formal Security Policy Model.* Computer Science Laboratory, SRI International, July, 1987.

17. Shockley, W.R., and Schell, R.R. *TCB Subsets for Incremental Evaluation.* In **Proc. of AIAA/ASIS/IEEE 3rd Aeorspace Computer Security Conference.** 1987.

18. Hinke, T.H. and Schaefer, M., *Secure Data Management System.* RADC-TR-75-266, System Development Corporation, Nov., 1975.

19. Lunt, T.F., D.E.Denning, D.E., R.R.Schell, R.R., Heckman, M. and Shockley, W.R. *Element Level Classification with A1 Assurance.* **Computers and Security.** Aug. 1988.

20. T.F.Lunt, R.R.Schell, W.R.Shockley, M.Heckman and Warren, D. *A Near-Term Design for the SeaView Multilevel Database System.* In **Proc. 1988 Symposium on Security and Privacy.** 1988.

21. Harrison, M. A., Ruzzo, W. L, and Ullman, J. D., *Protection in Operating Systems*, **Commun. A. C. M.,** Vol. 19, p. 461, 1976.

22. Schroeder, M.D., and Saltzer, J.H. *A Hardware Architecture for Implementing Protection Rings*, **Commun. A. C. M.,** Vol 15, p 115, 1972.

23. Schaefer, M., and Schell, R. R., *Toward an Understanding of Extensible Architectures for Evaluated Trusted Computer Products*, In **Proc. 1984 Symposiom on Security and Privacy**, IEEE Computer Society, 1984.

24. Reed, D., and Kanodia, R. K., *Synchronization with Eventcounts and Sequencers.* **Commun. A. C. M.,** Vol. 22, p. 115, 1979.

25. Schell, R.R., and Irvine, C.E., *Performance Implications for Multilevel Database Systems.* In **Proc. Third RADC Workshop on Multilevel Database Security,** MTP-385, MITRE, 1990.

EVALUATION BY PARTS OF TRUSTED DATABASE MANAGEMENT SYSTEMS

Ravi S. Sandhu

Center for Secure Information Systems
&
Department of Information and Software Systems Engineering
George Mason University, Fairfax, VA 22030-4444

1 INTRODUCTION

A database management system (DBMS) is a superb tool for building effective information systems. The widespread use of DBMS's across the board, from stand-alone personal computers at one end to heterogeneous networked mainframes at the other, is ample testimony to the success and acceptance of this technology. It is therefore no surprise that there is significant interest in trusted DBMS's on the part of users, vendors and evaluators.

The vast majority of DBMS's are hosted on top of some general-purpose Operating System (OS). The open systems thrust which is driving the industry will increasingly lead to situations where the DBMS, OS and possibly hardware are supplied by different vendors. This presents a significant challenge to vendors and evaluators in the development and rating of products.

Over the past few years, the security community has spent considerable effort in examining how the evaluation of a trusted DBMS can be factored out and separated from the evaluation of the underlying trusted OS. This effort has culminated in the recently published Trusted Database Interpretation (TDI) [1] of the Trusted Computer Security Evaluation Criteria (popularly known as the Orange Book) [2].

The authors of the TDI have consciously taken a generic (i.e., non-DBMS specific) and conservative approach to the question of evaluation by parts. This is perhaps appropriate in the first cut at this problem. However, an abstract and generic treatment leads to worst-case scenarios and pessimistic conclusions.

The authors of the TDI have also taken the approach that the Orange Book is a fixed parameter in this exercise. There has been no debate on whether or not the Orange Book could be strengthened or augmented in some way, so as to facilitate evaluation of parts.

In this note we propose some ideas on how the pessimistic conclusions of the TDI can be relaxed. We also speculate on some ways in which the Orange Book might be augmented to make evaluation by parts easier. There are no definite conclusions reached in this note. Our objective is to point out some avenues for fruitful research in the theory and practise of evaluation by parts.

2 EVALUATION BY PARTS

The writers of the TDI have deliberately chosen to take an abstract and (mostly) non-DBMS specific approach to the generic question of evaluation by parts. For instance the TDI states quite explicitly that:

> "The approach taken in this document is to address the issues of evaluating systems built of parts in a way that is independent of the field of trusted database management. This conscious attitude of generality is intended to make clear the distinction between the larger system-of-parts issues and the more specific DBMS issues."

While there are many merits to this approach, the general abstract setting inevitably leads to very conservative and cautious conclusions regarding the efficacy of evaluation by parts.

In particular the following pessimistic conclusion has dismayed many vendors, evaluators and researchers.*

> "This case also concerns a TCB that consists of two candidate TCB subsets, C and D. C is the more primitive subset. That is, D uses the abstractions provided by C .... Additionally, D is trusted with respect to C. That is, some of the C-subjects which make up TCB subset D execute as trusted processes of C. ... This case can be viewed as a special case of a previously evaluated TCB which has been altered. ... Although this case may appear, intuitively, to be different from that of arbitrary alteration of a previously evaluated TCB, the example demonstrates that such an approach makes it impossible to perform an evaluation by parts."

In this case the TDI is clearly treating the addition of a trusted process as an "arbitrary alteration of a previously evaluated TCB." Hence the "impossibility" of evaluation by parts.

The reality of system building and software engineering is, however, very different from this hypothetical worst case scenario. Trusted subjects are used to do very specific tasks and require very specific exemptions from the security policy of the underlying TCB.

*This conclusion was cast in different words in drafts leading to the eventual publication of the TDI as follows: "Thus, a TCB that contains an unconstrained TCB subset will be subject to the full gamut of penetration testing and covert channel analysis appropriate to the target evaluation class for the entire TCB, including any previously or separately evaluated TCB subsets. In the case of TCB subsets less primitive than an unconstrained TCB subset, only local evaluation activities can be done separately." Operationally, the net effect of the revised statement is essentially the same as this earlier formulation.

It is very important for the security community to examine the extent to which the conservative conclusions of the TDI can be relaxed in the light of specific trusted DBMS and trusted OS architectures? We are motivated by the following intuition: the extent to which the underlying trusted OS needs reevaluation should correlate with the degree of match between DBMS and OS architectures. We must go beyond the generic abstract perspective of the TDI to consideration of:

- concrete and realistic trusted DBMS and OS architectures, and
- allocation of overall security policy to individual DBMS and OS TCB subsets.

It is our conjecture that:

- the extent to which the underlying trusted OS needs reevaluation should correlate with the degree of match between DBMS needs and OS services, and
- the extent to which the underlying trusted OS needs reevaluation should correlate with the allocation of the overall security policy to individual DBMS and OS TCB subsets.

3 EXAMPLES

It is worth considering some examples to illustrate why the DBMS architecture and OS architecture are both relevant to this question.

1. Our first example shows the relevance of the OS architecture, specifically concerning the granularity of privilege that can be assigned to a trusted subject. At one extreme let OS-A provide only a binary privilege (such as super-user) for this purpose so that granting the privilege removes all constraints from the trusted subject, i.e., OS data structures are completely exposed to trusted subjects. At the other extreme let OS-B have extremely fine grained privileges to the extent that exemption from the $\star$-property can be granted with respect to individual files. It is intuitively obvious that an unconstrained TCB subset built on top of OS-A should require a greater degree of global penetration testing than one built on top of OS-B with one very specific exemption from the $\star$-property.

2. Next consider the relevance of the DBMS architecture. At one extreme consider DBMS-A which runs as a single trusted subject with respect to the OS. By definition such a DBMS has the ability to leak its highest sensitivity information to its lowest clearance subjects at essentially instantaneous speed. At the other extreme let DBMS-B run as a collection of untrusted processes, one at each sensitivity level, except for a single trusted process which is used to synchronize

the two-phase commit of transactions. It intuitively appears that DBMS-A should require greater degree of global covert-channel analysis than DBMS-B.

3. Finally consider the relevance of the combination of DBMS and OS architectures. Our intuition suggests that the 4 combinations resulting from coupling the 2 OS's and 2 DBMS's outlined above can be ranked as follows in decreasing order of global analysis required.

   (a) DBMS-A, OS-A

   (b) DBMS-B, OS-A

   (c) DBMS-A, OS-B

   (d) DBMS-B, OS-B

   One would expect an order of magnitude difference in effort required at the two extreme ends of this scale.

The challenge is to make a systematic analysis of various architectures to give a sound foundation for the intuitive ideas discussed above.

The compatibility of DBMS and OS architectures is strongly influenced by allocation of the overall security policy to individual TCB subsets (i.e., the DBMS and OS subsets). In reality the DBMS subset is itself likely to be factored into more than one TCB subset.[†] The allocation of policy to the individual TCB subsets is then a critical factor in determining the overall efficacy of evaluation by parts. The conservative conclusions of the TDI are important and significant because they do not depend upon specific assumptions in this regard. But then they are also overly conservative with respect to particular policy allocations.

4 OS REQUIREMENTS

The TDI is of course an interpretation of the Orange Book. It therefore takes the Orange Book as a given. The Orange Book on the other hand was not written with the concept of evaluation of parts in mind. Therefore the Orange Book does not evaluate the features of an OS which make it easy or hard to build trusted subject DBMS's on top. As our examples above show the privilege features of an OS are extremely relevant. We therefore need additional criteria so as to be able to make the following statement: It will be easier to incrementally evaluate trusted subject DBMS's built on top of OS A in comparison to those built on top of OS B. This is a difficult problem but one that must be confronted if products are actually going to be built and used.

[†]This is true of practically all DBMS architectures proposed in the literature.

5 SUMMARY

We have outlined some ideas for the critical and systematic analysis of evaluation requirements imposed by the coupling of various trusted DBMS and trusted OS architectures. The objective is to go beyond the conservative conclusions of the TDI by departing from its generic abstract perspective to consideration of

- concrete and realistic DBMS and OS architectures,

as well as

- specific assumptions regarding the allocation of the overall security policy to individual DBMS and OS TCB subsets.

We have also argued that the security community needs criteria to determine which OS architectures are more amenable to incremental evaluation of trusted-subject applications.

References

[1] National Computer Security Center. *Trusted Database Interpretation of the Trusted Computer Systems Evaluation Criteria.* NCSC-TG-021 (April 1991).

[2] Department of Defense. *Department of Defense Trusted Computer Systems Evaluation Criteria.* DOD 5200.28-STD (December 1985).

Metadata Issues in a MLS DBMS

Dick O'Brien
SCTC
1210 W. County Road E Arden Hills, MN 55112
Phone: 612-482-7443
Email: obrien@sctc.com

A relational DBMS usually includes two tables used to maintain the metadata: the Relation and Attribute tables. Users can perform queries on these relations just like any other relation in the database. Modification of the metadata is done by special operations such as *Create Relation*, *Modify Relation*, and *Destroy Relation*. These special operations insure that both the Relation and Attribute tables are modified consistently.

In a multilevel RDBMS there are a number of security issues that must be addressed when managing and designing the metadata. In this paper some of these issues are presented and discussed.

Destroying Relations

To simplify the discussion in this section, it is assumed that all of the metadata for a given relation must be defined at the same level. It may be possible that different relations are defined at different levels, but it is assumed that no relations are polyinstantiated. Whether or not polyinstantiation of tuples or elements is allowed in the relation is not important to the discussion in this section. So for simplicity, we assume no polyinstantiated data.

Since a relation may have tuples in it at numerous levels, the issue arises as to the manner in which a relation can be destroyed securely. If a user at the level at which the relation metadata is declared has the ability to destroy the relation, then higher level tuples in the relation will either be destroyed in the process or left as orphans. In either case, the potential exists for a low level user to make higher level data unavailable to high level users. While no high level data is disclosed by such an action, this does allow a denial of service attack.

Possible alternatives to allowing arbitrary users to delete relations would be to disallow the destruction of a relation that contained tuples at a higher level, to automatically redefine the metadata for the relation at a higher level, or to administratively control the manner in which relations are destroyed.

Disallowing the destruction of a relation if it contains any tuples at a higher level leads to a well-known covert channel in which a high level subject can signal information to a low level user by entering or removing data from the relation. If the relation is destroyed, then the low level user knows that there was no high level data in it. This approach may or may not be acceptable depending on the ease with which users (or user's subjects) can create and destroy relations. At any rate, the channel could easily be audited and the bandwidth controlled.

Automatically redefining the metadata at a higher level is a potentially complicated solution that is probably not feasible. In cases where there is data at higher levels which are incomparable with one another, the metadata would have to be redefined at each of these incomparable levels. This would lead to polyinstantiated relations and the resulting complications that they would imply.

The administrative approach seems the most feasible and could be handled in at least two ways. The simplest technique would be to have a database administrator who was cleared to database high. This administrator would have the responsibility of destroying relations and, when doing so, of ensuring that no higher level data is lost that should not be lost. A second technique, which would involve more functionality in the DBMS, would be to use a locking mechanism to keep relations from being destroyed if they had any higher level tuples in them. Before a user inserts tuples into a relation at a higher level, the metadata must first be locked by a user at the lower level. Before locked metadata can be manipulated at the lower level, the individual making the changes must notify individuals at the higher level of the changes and receive acknowledgment that the change is acceptable. Certainly individuals at the lower level can infer that there may be higher level data using this approach, but that is all they can infer. This may seem like a complex administrative procedure, but in fact when a relation is defined it can be locked immediately if it is known that the higher level users will be inserting data into it.

Classifying Metadata

There are several classification issues that must be addressed when deciding how metadata is to be handled in a MLS DBMS. Throughout the discussion, the emphasis will be on approaches that avoid polyinstantiation of metadata.

- The existence of relations may be classified.

  In certain applications it may be desirable to hide the existence of a higher level relation from lower levels. This would involve entering and maintaining the metadata for the relation at the higher level. As with ordinary data this leads to a potential covert channel. The channel occurs when a lower level user attempts to define a relation that already exists at the higher level. If the lower level user is notified of the conflict, a channel is created. This covert channel in metadata manipulation may be acceptable depending on how the database is used. If modifying the metadata occurs infrequently, or is only performed by a database administrator who is cleared to database high, the bandwidth of the channel is low, or non-existent. If database processes are allowed to create their own shared relations, however, name conflicts across levels might occur. To avoid polyinstantiating the relations, an appropriate solution might be to require that relations defined at different levels be named in such a manner as to identify the level at which they are defined. For example, TS_Ships and UNC_Ships. There need not, of course, be any relationship between tuples in the two relations since the relations might represent totally different real world entities.

- The existence of attributes may be classified.

  In certain applications it may be desirable to hide the existence of an attribute in a relation at lower levels. A name constraint could be placed on the attribute to require that all values for that attribute be stored at the higher level. For example, such a constraint might be: the destination of all ships must be classified Secret or above. However, using this approach, the existence of the "destination" attribute is still known at the lower level. If the existence of the attribute must be classified, then some other approach must be used.

The most straightforward approach is to require that all metadata that describes a relation be stored at a single level and then construct views that contain relations at different levels. A relation is not visible below the level of its metadata, nor are there attributes associated with the relation at higher levels. Instead attributes whose existence is classified are simulated by creating a new relation at the higher level that has the same key as the original relation and includes the classified attribute. A database view can then be defined at the higher level that joins the original relation with the higher level relation. This database view becomes the relation for the higher level.

The advantage of this approach is that the existence of attributes and relations can be hidden while keeping the metadata simple. The complete relation can be seen as the join of the lower level relation with the higher level relation. Since this view is defined as needed, a problem does not arise when incomparable levels have attributes with the same name. For levels dominating both the incomparable levels, the view can be defined using the desired combination of attributes. Since views and relations share the same name space, however, the view must have a different name.

For example, if the unclassified Ship relation is extended at the secret level with the attribute "Weapon", then the metadata might be;

Relation (U)

Relation Name	Arity	...
Ship	4	...

Relation (S)

Relation Name	Arity	...
Arms	2	...

Attribute (U)

Attribute Name	**Relation Name**
Ship Name	Ship
Class	Ship
Weight	Ship
Crew Number	Ship

Attribute (S)

Attribute Name	**Relation Name**
Ship Name	Arms
Weapon	Arms

View (S)

View Name	View Definition
ArmedShip	Ship $\bowtie_{ShipName}$ Arms

There are disadvantages to this approach. Updating the classified attribute becomes more difficult because it is maintained in a separate relation. It must either be maintained in the separate relation or the ability to update simple views of this type must be provided by the DBMS.

If polyinstantiated tuples are allowed in the relations, then other problems also arise. With this view definition, tuples that are polyinstantiated in the relation Ship would also be polyinstantiated in the view ArmedShip. It would, however, be possible to define the view to only include tuples in the Ship relation that are at secret.

In general, there are large number of possible views that might be defined. If R_1 is the lower level (U) relation and R_2 the higher level (S), then there are three possible ways to define a view at the higher level; $R_1 \bowtie R_2$, $R_1(restrictedtoU) \bowtie R_2$ and $R_1(restrictedtoS) \bowtie R_2$. Adding a third attribute at a third level in a relation R_3 creates even more possibilities. This is similar to what happens with polyinstantiated data elements in a tuple and could lead to some of the same confusion that arises in element level polyinstantiation [1]. However, since the views are defined by a human, the inconsistency can be resolved in the view the human creates. This places more burden on the schema designer, but the benefit is that the view can be defined exactly as desired for each level. Views provide the semantic link that ties together the relations at different levels into one relation.

In fact this approach does not require that views be defined. Users could create their own view, or construct queries that make the appropriate joins. While this approach is simple to implement, the DBMS cannot enforce that the views are defined in a consistent manner. Thus ambiguities might arise, especially when polyinstantiation (either tuples or elements) is allowed, unless great care is taken by the individuals defining the views.

Other approaches that might be used to solve the problem of classifying the existence of attributes are discussed in [2].

- The existence of classification constraints may be classified.

  Classification constraints, by their very nature, are difficult to hide. If a constraint requires values to be inserted at a higher level, the user entering the tuple can infer the existence of the constraint when values entered disappear, or the update is denied. In addition, if a classification constraint is classified, then a trusted mechanism is necessary to enforce the constraint at lower levels since only a subject at or above the level of the constraint would know of its existence. Thus, classifying constraints at a level higher than the level of the metadata that describes the relation is probably of limited value and may result in more complication to the DBMS than is warranted.

The Value of Constraints

The question arises as to whether content constraints have sufficient practical value to warrant their inclusion in a MLS DBMS.

Two possible uses that have been identified are:

- To provide a convenient way for a user to enter a tuple that contains multilevel data without having to enter portions of the tuple at each appropriate level. This is probably not an appropriate use of classification constraints since it encourages users to enter higher level data at a lower level, and if the entire database is not trusted, the potential exists that a low level Trojan horse could intercept the higher level data before it is upgraded.
- To provide an automatic upgrade facility for raw data that is being entered into the database for the first time. Such data may have been generated by sensors or other automatic data gathering devices and the classification of the data may depend on its value. In these instances it might be appropriate to have constraints do the upgrading automatically.

I would be interested in hearing of other possible scenarios in which content constraints might be of value.

Summary

Handling metadata in an MLS DBMS presents additional challenges to the designer of such a system. This paper has described some of these concerns and has discussed some of the possible solutions that might be chosen.

References

[1] S. Jajodia and Ravi Sandhu. Polyinstantiation Integrity in Multilevel Relations. In *IEEE Symposium on Security and Privacy*, pages 104–115, Oakland, CA, May 1990.

[2] R. C. O'Brien, J. T. Haigh, and D. J. Thomsen. Trusted Database Consistency Policy. Technical Report RADC-TR-90-387, Rome Air Development Center, Griffiss AFB, New York, December 1990.

SECTION 5

APPLICATIONS DESIGN

The papers in this section address issues relating to the design of multilevel database applications that enforce appropriate application security policies. In " 'Inference' Free Multilevel Databases," T.Y. Lin analyzes the security properties of relational operations when combining data classified at different security levels. He applies techniques for classifying aggregations to show that these techniques also eliminate inferences derived from the use of relational algebra. In her paper "Handling Security Constraints During Multilevel Database Design," Bhavani M. Thuraisingham provides an algorithm for relational decomposition based on association constraints, where attributes are clustered into uniformly classified groups. Thuraisingham, Ford, and Collins outline an integrated approach to security constraint processing in their paper "A Note on Security Constraint Processing in a Multilevel Secure Database Management System." They propose addressing security constraints at three distinct points in time: during database design, on each database update, and during query processing. In her paper "The Multipolicy Model: a Working Paper," Hilary H. Hosmer addresses the problem of integrating different, possibly conflicting, policies into a coherent security policy for a complex system. She describes the current paradigms, pointing out the inherent problems, and proposes a set of goals and mechanisms for a more flexible approach.

"INFERENCE" FREE MULTILEVEL DATABASES

T. Y. Lin

Mathematics and Computer Science
San Jose State University
San Jose, California 95192

1. INTRODUCTION

Inference is usually defined as a logical process of proving or deriving some conclusions from some given facts and rules [Morg87,88]. However, in the database security community, the term is used in a more technical sense, namely, the process of drawing a sensitive conclusion from joining (a relational operator) several less sensitive relations [Hink89], [Lunt89]. In this paper we will use the "inference" in this narrow sense.

Let E, F, G and H be relations and

$$E = F * G * H.$$

If E is strictly more sensitive than F, G and H, then there is an "inference". In general, suppose there is a relational algebraic equation

$$E = f(F, G, H, \ldots)$$

where E, F, G , H and ... are relations. Further, if E is strictly more sensitive than each F, G, H and ..., then we say there is an "inference". Intutively, it means we can draw a piece of senstive information E from less sensitive relations F, G, H ... and algebraic expression f. Intuitively speaking, the "inference" exists because the relation E is improperly classified.

Formaly, a database is "inference" free if

$$[E] =< \text{l.u.b} \{[F],[G],[H],..\}$$

is true for any relational expression E = f(F, G, H,...).

The central question then is:

Can we properly classify the data in a database so that such "inference" does not exit ?

There is an easy solution. If the data of a database are all classified Top Secret, then such "inference" can not exist. (For such situations, even the more general inference does not exist).

Obviously, such a security policy is not acceptable to almost any organization. Recall that a security map is the assignment of each object or subject to a security classification (or clearance). So the central problem is:

What is the "minimal" constraints on the security map so that the database will be "inference free"?

In this paper, we have pleasant results: We show that therе is no "inference" in lattice model and aggregated security algbera. Lattice modelis an aggregated security algebra when there is no aggregation problem in databases.

2. MULTILEVEL RELATIONAL DATABASES

From a structural point of view, a database is a collection D of structured data. In fact a database is a collection of data generated by relational operators. From a functional point of view, a database is a repository of information. A piece of information is a sub-collection (a view) of primitive data. Algebraically it is a "homogneous polynomial" of primitive data. We will use the set of all views, the set of information, the power set P(D), and the set of all homogeneous polynomial interchangeably.

2.1 Database without Aggregation

Intrinsically, a database can be viewed as a relational algebra generated by primitive data via relational operators [Lin89]. We will illustrate it by examples.

Example 2.1. Given the following relation E, we will represent it algebraically via relational operators.

E = US-TROOPS

city_name	Cl	#_troops	CL	Tuple_class	.....
Rome	S	755	S	S	
Athens	S	345	S	S	
London	C	231	C	C	......
Berlin	S	500	S	S	
Paris	S	500	S	S	
TOTAL	U	2331	U	U	

This relation can be expressed algebraically as follows:

```
    E =  Rome X  755  U Athens X 345
      U  London X 231 ....
```

where X is the cartesian product and U is the union. Rome X 755 is the first tuple in E (without classification)

Remark: If one think of Rome, 755, Athens,... as attibute value pairs (supressing the notation of attribute), then E can be written as

E = (Rome)(755)+(Athens)(345)+(London)(231)+...

where the record is represented by juxtaposition and union by +. This is a "formal homogeneous polynomial" on literals.

To compute the tuple security class, we can computed as follows:

[E] = [Rome X 755] # [Athens X 345]
    # [London X 231] ....

where # is the least upper bound operator and [Z] denotes the security class of Z.

If the relation has been classified to the element level, then we can re-express it by

[E] = [Rome] # [755] # [Athens] # [345]
    # [London] # [231] ....

Intrinsically, this a homomorphism from the ring of "formal homogeneous polynomials" to lattice (as a semi group of least upperbound operator), mapping both"plus" and "product" to the least upper bound operator.

We would like to emphasize that any piece of information in multilevel database should have a security classification (SC). In fact, if we interpret a multilevel database system by the Bell Lapdula model (such as SeaView), then indeed every piece of information should have a security classification, including items in data dictionary. Mathematically, there should be a security map from P(D) to SC.

[]: P(D) -----> SC

So a multilevel database system is a triple

(P(D), [], SC),

where D is all the primitive data including items in data dictionary.

In almost all of the works in computer security, all have assumed that a security map on primitive data is given

[] : D -------> SC

As we have mentioned earlier that the security map [] should have an extension to the power set P(D). This extension is an exponential problem. In [Lin90], we propose to extend the security map algebraically.

Let us rephrase the Example 2.1, the primitive data is

```
D = {Rome, 755, Athens, 345, London,  231,  Berlin, 500
      Paris,  500,  TOTAL,  2331}
```

and their security classes are assigned in the table, for example [Rome]=S, [755]=S.

Here a tuple is a de~~ived data, since it is a cartesian product of elements. The tuple classes can be computed by the least upper bound operator. The relation also is a dervied data since it is union of tuples. All their security classes can be computed by least upper bound operator.

As examples, let us compute the security class of E:

```
        [E] =  [Rome] # [755] # [Athens] # [345]
            #  [London] # [231] ....
            = S # S # S # S
            # C # C .....
            = S.
```

So the classification of E is S.

2.2. Database with Aggregation

Now, we turn our attention to the case in which database has aggregation problem. Roughly speaking, a collection of facts has a classification strictly greater than that of the individual facts forming the aggregate. Moreover, as Lunt remarked, "To qualify as an aggregation problem, it must be the case that the aggregate class strictly dominates the class of every subset of the aggregate." We refer to [Lin89c], or [Lin90] for formal treatments.

A solution of a single aggregation is:

Let E = {E1,E2,...Ek} be an aggregate

(1) Add to the database a new primitive generator E
(2) Remove E1, E2,...Ek from the original primitive generators
(3) The release of any of E1, E2,..., Ek are handled by SSO or a special mechanism [Lin90]. Effectivel, Ei's are "upgraded" to the level of aggregate E.

A solution for the aggregation problem is an algorithm called APR [Lin90], which selects a subset of aggregates called marked aggregates. The family of marked aggregates has the following properties

(1) No compromise in protection (provide the same protection as original aggregates)
(2) If the SC is linear, then they are pairwise disjoint. (each data belongs to at most one aggregate; in any real implementation, one needs this)
(3) If [E] and [F] are non comparable, they E and F may have intersection.

Example 2.2

Let E' be the same as above example E except that the security classification of the first tuple is changed to

E'

city_name	Cl	#_troops	CL	Tuple_class	.....
Rome	S	755	S	TS	

(rest of the items are same as above)

That is, the first tuple class is TS which is strictly higher than [Rome] and [755]. So this tuple is an aggregate.

We recall that if the database has aggregation, then the primitive data D is the collection of

(1) marked aggregates and
(2) primitived data that are not in any marked aggregate.

For this example, we have the following primitive data

(1) {Rome, 755} -- marked aggregate
(2) Athens, 345, London, 231, Berlin, 500
Paris, 500, TOTAL, 2331

Note the security class of {Rome, 755} is TS.

Let us compute the security class of the following view (instance):

V:

city_name	Cl	#_troops	CL	Tuple_class	.....
Rome	S	755	S	TS	
Athens	S	345	S	S	
London	C	231	C	C	......

The security class of this view V can be computed as follows:

```
[V] = [Aggregate containing Rome] # [Athens] # [345]
    # [London] # [231] ....
    = [Rome X 755] # [Athens] # [345]
    # [London] # [231] ....
```

```
= TS # S # S
# C # C .....
= TS.
```

Let us consider another view W

W

```
    city_name
    ---------
     Rome
     Athens
     London
     Berlin
     Paris
```

Then the security class of this view is:

```
[W] =  [Aggregate Containing Rome] # [Athens]
    #  [London] # [Berlin] # [Paris]
    =  TS # S # S # C # S # S
```

By our aggregation solution, Rome is in the aggregate {Rome, 755}, so its security class is the class of [Rome X 755] = TS.

```
    = TS
```

Example 2.3

Let A be the follwoing sub-relation of E in Example 2.1

city_name	Cl	#_troops	CL	Tuple_class	.....
Rome	S	755	S	S	
Athens	S	345	S	S	
London	C				

The classification of A is [A] = TS. Since [A] is strictly higher than each individual elements and any subsets, it is an aggregation.

Let us consider a view (instance)

X:

city_name	Cl	#_troops	CL	Tuple_class	.....
Rome	S	755	S	TS	
Athens	S	345	S	S	
London	C	231	C	C	......

The security class of this view V can be computed as follows:

```
        [X] = [Aggregate containing Rome]
            # [Aggregate containing Athens]
            # [Aggregate containing 345]
            # [Aggregate containing London]
            # [231] ....
            = [A] # [A] # [A] # [231] ....
            = TS # TS # TS # TS # C
            = TS
```

3. LATTICE MODEL - AN "INFERENCE" FREE MULTILEVEL DATABASE

Let D be a primitive data of a database (there is no aggregation problem) and its security classes are assigned, say by DOD. Namely the map

$$[\,] : D \longrightarrow SC$$

is given by DOD.

The extension

$$[\,]: P(D) \longrightarrow SC$$

is given by the "homomorphism" as explained in Example 2.1.

Intuitively, it means every view (instance) is defined by the least upper bound of its primitive data. For example, the security class of a tuple (which is a view with one row) is the least upper bound of individual elements. For example, the security class of the tuple

[Rome X 755] = l.u.b of [Rome} and [755]

So the tuple class of Rome X 755 is S.

This is essentially Denning's lattice model [Denn76]. The security class of a derived data is the least upper bound of its component.

Let us examine the behavior of the security classes under relational operators.

Under lattice model, the security class of a collection of data is the l.u.b. of all individual elements (note this is a database without aggregation). So we have the following results:

(3.1) Cartesian product

[F X G] = [F] # [G]

F X G consists all possible elements in F and G, so [F X G] is the

[F X G] = [F] # [G]

F X G consists all possible elements in F and G, so [F X G] is the l.u.b. of all possible elements in F or G, hence it is the l.u.b. of [F] and [G]. This prove the assertion.

(3.2) Union

[F U G] = [F] # [G]

The proof for Cartesain product works for union too.

(3.3) Differecne

[F \ G] =< [F]

F \ G is a subset of F.

(3.4) Divide

[F / G] =< [F]

F / G is a subset of F.

(3.5) Intersection

[F ∩ G] =< g.l.b. ([F], [G]) = [F]^[G]

where ∩ is intersection and [ ]^[ ] is the g.l.b. in lattice.

(3.6) Selection

[Select F, where ....] =< [F]

A Selection of F is a subset of F.

(3.7) Join

[F * G] =< [F] # [G]

By definition, F * G is a subset of Cartesian product F X G.

so [F * G] =< [F X G] = [F] # [G]

(3.8) Projection

[Proj F to ....] =< [F]

A projection of F is a subset of F

These analysis leads us to conclude the follwoing proposition.

E = f(F, G, ...)

be an algebraic expression of relational algebra. Then

[E] =< l.u.b. {[F], [G],...}

Proof: By replacing all the relational operators in f by the right-most expression of (3.1)-(3.8), we have

[E]=[f(F,G,..)] = [f]([F].[G],..) =< l.u.b.{[F].[G],..}

where [f] is the corresponding algebraic expression of f in the lattice SC.

Verbally, the proposition concludes that

[Relational operations on F, G, ..] =< l.u.b. {[F],[G],..}

In other words, we have

Theorem. If the security classification of a database gives rise a lattice model, then the database is "inferecne" free.

This should not be a surprise, see Conclusion.

4. AGGREGATED SECURITY ALGEBRA - AN "INFERENCE" FREE MULTILEVEL DATABASE

Now let us consider the multilevel database with aggregation [Lin90a]. Again the security map []:D --> SC is given by DOD and the security map []: P(D) --> SC can be determined by the aggregated security algebra. Note that D consists of marked aggregates and primitive data which are not in any of the aggregates.

Once D is chosen, the aggregated security algebra behaves like lattice model. Therfore all the arguments in last section applies. Just for example, let us examine the case of join operator.

Let X be a relation. Then, in aggregated security algebra

[X] = the least upper bound of

(1) primitive data in X and
(2) aggregates which contains elements in X
(See the computation of Example 2.2 and 2.3).

As in (3.7) last section, F * G is a subset of Cartesian product F X G, so

[F * G] =< [F X G] = [F] # [G].

That is,

$$[E] =< [F] \# [G]$$

Therfore, the inference can not exist in aggregated security algebra either. Let us quote the reuslts without proof.

Proposition. Let

$$E = f(F, G, ...)$$

be an algebraic expression of relational algebra. Then

$$[E] =< \text{l.u.b.} \{[F], [G],...\}$$

Verbally, the proposition concludes that

$$[\text{Relational operations on } F, G, ..] =< \text{l.u.b.} \{[F],[G],..\}$$

In other words, we have

Theorem. If the security classification of a database gives rise an aggregated security algebra, then the database is "inferecne" free.

5. FORMAL SECURITY ALGEBRA

In implementing such computation, the notion of formal security agebra is beneficial. During the query processing, a query may go thoruh all the intermediate computations and hence reaching a security class that maybe well above the security class of final results (especailly if one uses the high-water-mark policy). To avoid this one should use the notion of formal security classes. Formal security classes form a ring of formal polynomials over the security classes as its formal variables. The formal classes are not interpreted until the final result. For example, binding the formal product with the leat upper bound operator of lattice will not be done until the output time. The actaul development of formal security algebra is quite lengthy and it may confuse the essential issues here. An exposition of formal security classes is too much a digression here, even though we did illustrate the idea in the oral presentation at workshop.

6. CONCLUSION

The results may surprise many readers. But it is a natural consequence of the theory of security algebra. Note that in aggregated security algebra or lattice mode, a derived data is assigned a security class by algebraic relation. So in the aggregated security algebra and lattice model, there is no inconsistency in security classification among variables or data in algebraic relations. The "inference" is dervied from the inconsistency of assigning the security classes among datas in algebraic relations. So our theorem is a natural consequence of security algebra. In literature, aggregation and "inference" problems are separated, in fact, from our point of view the two notions are in one. A good solution for aggregation problem always implies a solution on "inference" problems (that is, there is no "inference"; again we should disclaim here that there are no results on inference.

Acknowledgement
This research was partially supported by NSA(MDA904-91-C-7048) and RADC. We would like to thank Howard Stainer and Michael Ware, and Joseph Giodano for their support.

References

[Date86] C. J. Date, An introduction to Database Systems, Addison-Wesley, Reading, MA., 1986

[Denn76] D. E. Denning. "A Lattice Model of Secure Information Flow", Communications of the ACM, Vol. 19, No. 5, May 1976, pp. 236 - 243.

[Hink88] T. H. Hinke. Inference Aggregation Detection in Database Management Systems, Proceedings of the 1988 IEEE Symposium on Security and Privacy, April 1988.

[Lin89a] T. Y. Lin, A Generalized Information Flow Model and Role of System Security Officer, Database Security: Status and Prospects II, edited by C. E. Landwehr, North Holland, 1988.

[Lin89b] T. Y. Lin, L. Kerschberg, and R. Trueblood, Security Algebra and Formal Models, Proceedings of IFIP WG11.3 Workshop on Database Security September 5-7, 1989.

[Lin89c] T. Y. Lin, Commutative Security Algebra and Aggregation, Proceedings of Second RADC Workshop on Database Security, 1989.

[Lin90a] T. Y. Lin, Multilevel Database, Aggregated Security Algebra, Proceedings of IFIP WG11.3 Workshop on Database Security September 18-21, 1990.

[Lin90b] T. Y. Lin. Numercial Measure on Aggregations, Proceeding of 6th annual Application Computer security conference, Dec 3-7, 1990.

[Lunt89] T. F. Lunt. Aggregation and Inference: Facts and Fallacies, Proceedings of 1989 IEEE Symposium on Security and Privacy, 1989.

[Morg87] Mathew Morgenstern. "Security and Interference in Multilevel Database and Knowledge-base Systems," ACM International Conference on Management of Data (SIGMOD-87), May 1987.

[Morg88] Mathew Morgenstern. "Controlling Logical Inference in Multilevel Database Systems, Proceedings of 1988 IEEE Symposium on Security and Priv[illegible] 1988.

HANDLING SECURITY CONSTRAINTS DURING MULTILEVEL DATABASE DESIGN

Bhavani Thuraisingham

The MITRE Corporation, Burlington Road, Bedford, MA

Abstract

In this paper we describe techniques for processing association-based constraints, simple constraints, and logical constraints during multilevel database design.

1 INTRODUCTION

In a multilevel secure database management system (MLS/DBMS) users cleared at different security levels access and share a database consisting of data at different sensitivity levels. A powerful and dynamic approach to assigning sensitivity levels, also called security levels, to data is one which utilizes security constraints or classification rules. Security constraints provide an effective and versatile classification policy. They can be used to assign security levels to the data depending on their content and the context in which the data is displayed. They can also be used to dynamically reclassify the data. In other words, the security constraints are essential for describing multilevel applications.

Handling security constraints in multilevel database systems has received some attention during the past five years. An early attempt to identify various types of security constraints was made in [DENN86, DWYE87]. The security constraints that were identified included those that classify data based on content, context, aggregation, and time. Later, in an article [STAC90], we described the design of an MLS/DBMS in which security constraint processing was fundamental. The work reported in [THUR87, KEEF89, THUR89, THUR90a] suggest ways of handling security constraints during query processing in such a way that certain security violations via inference does not occur[1]. The work reported in [MORG87, HINK88, LUNT89, SMIT90] focuses on handling constraints during database design where suggestions for database design tools are given. They expect that security constraints during database design are handled in such a way that security violations cannot occur. While the previous papers discuss how security constraints may be handled, they do not provide viable approaches or algorithms for actually processing them.

From an analysis of the various types of security constraints, we believe that they are a form of integrity constraints enforced in an MLS/DBMS. This is because in a multilevel database one can regard the security level of an entity to be part of the value of that entity. Therefore security constraints specify permissible values that an entity can take. Since security constraints can be regarded as a form of integrity constraints, many of the techniques developed

[1] Inference is the process of drawing conclusion from premises. It becomes a problem if the conclusions drawn are unauthorized.

for handling integrity constraints in non-multilevel relational database systems by the logic programming researchers (see, for example, GALL78, MINK88]) could be used for handling security constraints in an MLS/DBMS. In these techniques, some integrity constraints, which are called derivation rules, are handled during query processing, some integrity constraints, known as integrity rules, are handled during database updates, and some integrity constraints, known as schema rules, are handled during database design. Our approach to handling security constraints has been influenced by (i) the approach taken to process integrity constraints by the logic programming researchers [LLOY87], and (ii) the design of Lock Data Views [STAC90], a high assurance MLS/DBMS, in which constraint processing is fundamental.

We have defined various types of security constraints. They include the following:

(i) Constraints that classify a database, relation or an attribute. These constraints are called simple constraints.
(ii) Constraints that classify any part of the database depending on the value of some data. These constraints are called content-based constraints.
(iii) Constraints that classify any part of the database depending on the occurrence of some real-world event. These constraints are called event-based constraints.
(iv) Constraints that classify associations between data (such as tuples, attributes, elements, etc.). These constraints are called association-based constraints.
(v) Constraints that classify any part of the database depending on the information that has been previously released. These constraints are called release-based constraints.
(vi) Constraints that classify collections of data. These constraints are called aggregate constraints.
(vii) Constraints that classify any part of the database depending on the security level of some data. These constraints are called level-based constraints.
(viii)Constraints which assign fuzzy values to their classifications. These are called fuzzy constraints.
(ix) Constraints which specify implications. These are called logical constraints. (Note that logical constraints do not classify any data and therefore cannot be regarded as security constraints. Nevertheless we will see that they play an important role in security constraint processing.)

We believe that an integrated approach to security constraint processing is necessary. That is, some constraints should be processed during the query operation, some during the database update operation, and some during the database design operation. While our previous papers (see, for example, FORD90, COLL90]) have described prototypes which process security constraints during the database query and update operations, this paper describes the techniques that we have designed to process security constraints during the database design operation. Our main focus is on processing the association-based constraints which classify relationships between a collection of attributes at a particular security level. We also describe briefly how simple and logical constraints could be processed during the database design operation.

The organization of this paper is as follows. In section 2 we describe techniques for handling the association-based constraints. A note on processing simple constraints is discussed in section 3. Handling logical constraints is discussed in section 4. The paper is concluded in section 5.

2. HANDLING ASSOCIATION-BASED CONSTRAINTS

2.1 OVERVIEW

An association-based constraint classifies a collection of attributes taken together at a particular security level. What is interesting about the association-based constraint is that it can generate several relationships between the various attributes. For example, if there is a relation SHIPS whose attributes are S#, SNAME, and CAPTAIN, and if an association-based constraint classifies the SNAME and CAPTAIN taken together at the Secret level, then one of the pairs (S#, SNAME), (S#, CAPTAIN) should also be classified at the Secret level. Otherwise, an Unclassified user can obtain the (S#, SNAME) and the (S#, CAPTAIN) pairs and infer the Secret association (SNAME, CAPTAIN). There has been much discussion in the literature as to the appropriate place to handle these association-based constraints. Some argue that they should be handled during database design [LUNT89] while others argue that they should be handled during query and update processing [STAC90]. However, none of the work reported so far studied the properties of these association-based constraints, nor has it provided any technique to generate the additional association-based constraints that can be deduced from an initial set of association-based constraints.

We first describe an algorithm which processes a given set of association-based constraints and outputs the schema for the multilevel database. Given a set of association-based constraints and an initial schema, the algorithm will output clusters of attributes and the security level of each cluster. We then prove that the attributes within a cluster can be stored securely at the corresponding level. A tool based on this algorithm can help the systems security officer (SSO) design the multilevel database. The algorithm that we have designed does not necessarily have to be executed during database design only. It can also be executed during query processing. That is, the query processor can examine the attributes in the various clusters generated by the algorithm and then determine which information has to be released to the users. For example, if the algorithm places the attributes A1, A2 in cluster 1 at level L, and the attributes A3, A4 in cluster 2 at level L, then, after an attribute in cluster 1 has been released to a user at level L, none of the attributes in cluster 2 can be released to users at level L.

Next, we study the properties of association-based constraints by treating them as a form of data dependency for multilevel databases.[2] We define the relationships that are generated between the various attributes from a set of association-based constraints as association dependencies. We then investigate ways of generating all the association-dependencies from a given set of association-based constraints. Our ultimate goal here is to generate a minimal derivation of the association dependencies. Using such a derivation, the relationships between the attributes could be assigned security levels which do not result in overclassification.[3]

[2] Data dependencies have played an important role in the design of ordinary (i.e., non-multilevel) databases [MAIE83].

[3] The intent is to assign minimum security levels to the various relationships between the attributes necessary to maintain security.

In section 2.2 we describe an algorithm for handling association-based constraints during database design. A note on association dependencies is discussed in section 2.3.

2.2 ALGORIHM FOR HANDLING ASSOCIATION-BASED CONSTRAINTS

In this section we describe an algorithm for handling association-based constraints. The input to this algorithm is a set of association-based constraints and a set of attributes. The output of this algorithm is a set of clusters for each security level. Each cluster for a security level L will have a collection of attributes that can be safely classified at the level L. That is, if A1, A2, and A4 are attributes in a cluster C at level Secret, then the attributes A1, A2, and A3 can be classified together safely at the security level Secret without violating security. The clusters are formed depending on the association-based constraints which are input to the program. Once the clusters are formed, then the database can be defined according to the functional and multivalued dependencies that are enforced.

ALGORITHM HABC (Handling Association-Based Constraints)

```
Begin

    Let C be the set of association-based security constraints and W1, W2, .......Wm be the
    set of attributes which are input to the program.

    An association-based security constraint is represented by A1•A2•..........•An = L where
    A1, A2, .....An are attribute and L is a security level. What it means is that the attributes
    A1, A2, .....An taken together must be classified at level L.

    For each security level L, do the following:

      Begin
            Let C[L] be the largest subset of C. That is, C[L] consists of the set of
            constraints which are visible at level L.

            Let A = {A1, A2, ......An} be the largest subset of {W1, W2, .....Wm}, such
            that the elements of A are visible at level L.

            Since n is the number of attributes which are visible at level L, clusters
            C1, C2, .....Cn will be formed as follows:

            Set C1 = C2 = C3 = .......= Cn = Empty-set.
            For each i ($1 \leq i \leq n$) do the following:

                  Begin
                        Find the first cluster Cj ($1 \leq j \leq n$) such that Ai, together with any
                        of the attributes already in Cj, are classified at a level dominated
                        by L by the set of constraints C[L].
```

```
                Place Ai in the cluster Cj. (Note that since we have defined n
                clusters, there will definitely be one such Cj.)
            End (for each i).

        Output all the non-empty clusters along with the security level L.

    End (for each security level L).

End (HABL)
```

Theorem 1: Algorithm HABL is Sound.

Proof of Theorem 1:

We need to show that for every security level L, the attributes in a cluster formed at L can safely be stored together in a file at level L.

Let C be a cluster at level L, and let B1, B2, ...Br be the attributes in C. Note that before each Bi is placed, it will be first checked to determine whether or not there is an association-based constraint which classifies Bi together with any subset of the attributes B1, B2, ...Bi-1 already in C at a level not dominated by L. If so, Bi would not have been placed in the cluster C.

Since this is true for each Bi ($1 \leq i \leq r$), there is no association-based constraint which classifies any subset of B1, B2, ...Br taken together at a level not dominated by L. Therefore, B1, B2, ...Br can be safely stored in a file at level L.

Theorem 2: Algorithm HABL is Complete.

Proof of Theorem 2:

We need to show that, if Ci and Cj are two clusters at a level L, there are subsets A and B, respectively, of Ci and Cj, such that A and B cannot be stored together in a file at level L.

Let i < j. Then the cluster Ci appears before Cj, in the enumeration of the clusters formed at level L.

Suppose, on the contrary, that A and B do not exist. Consider an element X of cluster Cj. Since Ci is before Cj in the enumeration, before placing X in Cj, it would have been first checked to determine whether or not X can be placed in Ci. It would have been found that there was an association-based constraint which classifies X together with the attributes already in a subset P of Ci at a level not dominated by L. That is, the subset P and {X} of Ci and Cj respectively cannot be stored in a file at level L. That is, we have found two sets, A and B, which are subsets of Ci and Cj, respectively, which cannot be stored in a file at level L. This is a contradiction to our assumption.

We now trace the algorithm with a simp¹ example.

Let the attributes be A1, A2, A3, A4, A5. Let the constraints be the following:
CON1: A1 • A2 = Secret[4]
CON2: A1 • A5 = Secret
CON3: A1 • A4 • A5 = Secret
CON4: A2 • A4 = Secret
CON5: A3 • A4 = Secret

Note that some of the constraints are redundant. For example, CON2 implies CON3. In this paper we are not concerned with the redundancy of the constraints.

Since the maximum classification level assigned is Secret, all the attributes can be stored in a file at the level Secret or Higher. At the Unclassified level, the following clusters are created:

C1 = {A1, A3}
C2 = {A2, A5}
C3 = {A4}.

It should be noted that, although the algorithm guarantees that the constraints are processed securely, it does not provide any guarantee that the attributes are not overclassified. More research needs to be done in order to develop an algorithm which does not overclassify an attribute more than is necessary.

2.3 A NOTE ON ASSOCIATION DEPENDENCIES

In this section we define a new type of dependency, which we call association dependency (AD) for multilevel databases. We then develop a set of inference rules which generates all association dependencies from the set of axioms. The axioms are the initial association dependencies which are given. Note that these initial dependencies are the association-based constraints that are specified. Our ultimate goal is to generate a minimal set of association dependencies from a set of association-based constraints. The minimal derivation will be such that it generates no more association dependencies than is necessary in order to provide security. A design based on such a derivation will guarantee that the attributes of the relations are not overclassified.

It should be noted that the first step toward the design of a multilevel database would be to decompose the attributes properly, based on the association dependencies. Once the association-based dependencies are handled, then further design of the database can proceed, depending on the functional and multivalued dependencies enforced. In this paper we do not address the issues of functional and multivalued dependencies for multilevel databases. We refer to the work reported in [DENN87, ONUE87, JAJO90] for a discussion on these other data dependencies.

[4] By "A1 • A2 = Secret" is meant: A1 and A2 taken together are classified at the Secret level.

Below we first define association dependencies and then describe some inference rules for generating association dependencies.

Association Dependencies

An association dependency is represented by (X1•X2•.......•Xn, L) where each Xi ($1 \leq i \leq n$) is an attribute structure and L is a security level. An attribute structure could be either an attribute or it could be a tuple of attribute structures. What the association dependency means is that the attribute structures X1, X2, ...Xn taken together is classified at the level L. It can be seen that every association-based constraint generates an association dependency. An association dependency will generate other association dependencies between the various attribute structures.

If there is an association-based constraint which classifies A1 and A2 taken together at a level L, then the corresponding association dependency will be denoted as follows: (A1 • A2, L).

Note that A1 and A2 do not have to be singleton attributes. They can be a tuple of attributes. That is, if there is a constraint which classifies the collection (X1, X2, ...Xn) together with the collection (Y1, Y2......Ym) at the level L, then the dependency is denoted by ((X1, X2, ...Xn) • (Y1, Y2, ...Ym), L).

For convenience, we can denote the tuple (X1, X2, ...Xn) by X and the tuple (Y1, Y2, ...Ym) by Y. Then the dependency is denoted by (X • Y, L).

Note that each association-based constraint can produce one or more association dependencies between the various attributes. For example, let an association-based constraint classify the attributes X and Y at level Secret. Let Z be a third attribute. Note that, if the attributes X, Z, taken together, are classified at the Unclassified level and the attributes Y, Z, taken together, are classified at the Unclassified level, then an Unclassified user could get the Secret association between X and Y. Therefore, at least one of (X,Z), (Y,Z) must be classified at the Secret level.

The question is, how can one enumerate all the association dependencies from an initial set of association dependencies? We answer this question by defining a set of inference rules for association dependencies which are sound and complete. That is, given a set of association dependencies, one can generate all the association dependencies using the inference rules. Further, any dependency that is generated is a valid dependency. Note that results along similar lines have been obtained for functional and multivalued dependencies [MAIE83].

Inference Rules

We state four inference rules to enumerate the association dependencies from an initial set of axioms. Each rule will be illustrated with examples. Issues on the consistency and completeness of the rules as well as the use of graphs for generating association-based dependencies are described in [THUR90b].

Rule 1: (Permutation Rule):

$\forall$ n ((X1 • X2 • ....Xn), L) ==> ((Y1 • Y2 • ....Yn), L)
where for each Y1, Y2, ...Yn is a permutation of X1, X2, ...Xn.

Example: If (A1 • A2, L), then (A2 • A1, L). That is, if there is an association dependency between A1 and A2 at level L (i.e., A1 and A2 taken together are classified at level L), then there is one between A2 and A1.

Rule 2: (Decomposition Rule)

$\forall$ n ((X1 • X2 • ....Xn), L)' ==> $\forall$ Z ((Y • Z, L) V ((X - Y) • Z), L)
where X = (X1, X2,........Xn) and Y is either a subtuple of X, or Y is a component of X.

Note: Y is a subtuple of X if Y = {Y1, Y2, ...Ym} and for each i ($1 \leq i \leq m$) Yi $\in$ {X1, X2, ...Xn}. X - Y consists of the remaining Xi's.

A component of X is defined as follows: Let UX be all the attributes which constitute X. Note that this is obtained by first finding all the attributes in each Xi and then forming the union of all of them. Y is a component of X if Y is a subset of UX. Then X - Y has all the elements in UX which are not elements of Y.

Example: (A1 • A2, L) ==> (A1 • A3, L) V (A2 • A3, L)

Rule 3: (Partitioning Rule) $\forall$ n ((X1 • X2 • ....Xn), L) ==>
((Y1 • Y2 • ....Ym), L)
where Y1, Y2, ...Ym is any m-tuple which constitutes X = (X1, X2,........Xn), Yi and Yj ($1 \leq i, j \leq m$) do not intersect and either Yi ($1 \leq i \leq m$) is a subtuple of X or Yi is a component of X.

Note that by Y constituting X we mean UY = UX.

Example: (A1 • A2 • A3 • A4, L) ==> (((A1, A3) • (A2, A4)), L)

Rule 4: (Augmentation Rule) $\forall$ n ((X1 • X2 • ....Xn), L) ==>
$\forall$ Z ((X1 • X2 • ....Xn • Z), L)

Example: (A1 • A2, L) ==> (A1 • A2 • A3, L)

We now define derivations of ADs from axioms. Given a set of ADs (which are regarded as the axioms), we say that an AD X • Y has a derivation if it is either an axiom or if it has a proof Z1, Z2, ...Zn such that Zn = X • Y; and for each i ($1 \leq i \leq n$), Zi is an axiom or it can be derived from {Z1, Z2, ...Zi-1} using the inference rules described in this section.

3. A NOTE ON SIMPLE CONSTRAINTS

Since simple constraints classify individual attributes at a certain security level, they could also be handled during database design. Note that when an attribute A in relation R is classified at level L, then all elements which belong to A is also classified at level L. Therefore, we can store A itself at level L.

The algorithm which handles simple constraint is straightforward. Each attribute that is classified by a simple constraint is stored at the level specified in the constraint. Once the algorithm for processing simple constraints is applied and the corresponding schema is obtained, then this schema is given as input to the algorithm handling association-based constraints. The association-based constraints are then applied and the final schema is obtained.

We illustrate the combined algorithm with an example.
Let relation R have attributes A1, A2, A3, and A4. Let the constraints enforced be the following:

Simple constraint: A4 is Secret
Association-based constraint: A2 and A3 together are TopSecret.

Applying the algorithm for handling simple constraints we obtain the following result.
A1, A2 and A3 are Unclassified; A1, A2, A3, and A4 are Secret.

Next we apply the algorithm for handling association-based constraints. The final output is:
A1 and A2 are Unclassified; A1, A2, and A4 are Secret; A1, A2, A3, and A4 are TopSecret.

4. HANDLING LOGICAL CONSTRAINTS

Logical constraints are rules that can be used to deduce new data from existing data. If a security constraint classifies the new data at a level that is higher than that of the existing data, then the existing data must be re-classified. Logical constraints could be straightforward such as Ai ==> Aj (which means that the value of Ai implies the value of Aj) or they could be more complex such as A1 & A2 & A3 & ....An ==> Am (which means that the values of A1, A2, ...An imply the value of An). In the case of the first rule, if Aj is classified at the Secret level then Ai must be classified at least at the Secret level. In the case of the second rule, if Am is classified at the Secret level, then at least one of A1, A2, ...An must be classified at least at the Secret level.

We have developed techniques to handle logical constraints during query processing as well as during database design. For example consider the constraint Ai ==> Aj. If Aj is classified at the Secret level, and an Unclassified user requests for Ai values, the query processor will ensure that the values for Ai are not released. That is, although Ai may be explicitly assigned the Unclassified level, since the logical constraint is treated as a derivation rule, it does not cause any inconsistency. That is, during query processing, the security level of Ai will be computed to be Secret.

For logical constraints which do not have any conditions attached, it appears that they could be handled during database design. That is during design time the logical constraints are examined, and the security levels of the attributes specified in the premise of a constraint could be computed. For example, if Aj is classified at the Secret level then it must be ensured during design time that Ai is classified at least at the Secret level also. The following algorithm will ensure that the security levels are computed correctly.

1. Do the following for each logical constraint. (Note that we have assumed that the constraints are expressed as horn clauses. That is, there is only one atom in the head of a clause.)

2. Check whether there are any simple constraints which classify the attribute appearing in the head of the logical constraint at any level. If not, ignore the constraint.

3. If so, find the security level L that is specified for this attribute.

4. Check whether any of the attributes appearing as premises of the logical constraint are classified at least at level L. If so, ignore the constraint.

5. If not, classify one of the attributes (say, the first one) at the level L.

The algorithm given above does not ensure that the attributes are not overclassified. In order to avoid the overclassification problem, modification must be made to step 5. That is, once an attribute is assigned a security level, it is possible for the level to be re-assigned based on other logical constraints that are handled. Our current research includes investigating techniques for successfully assigning security levels to the attributes and at the same time avoiding overclassification.

When logical, simple, and association-based constraints are combined, then the first step would be to handle the simple constraints. The next step would be to apply the algorithm given above for the logical constraints. Finally the algorithm given in section 2 is applied for the association-based constraints.

5. CONCLUSION

In this paper we first described association-based constraints and discussed the various approaches that have been suggested to handle them. Then we described an algorithm which generates the schema for a multilevel database when given a set of association-based constraints. We proved that the algorithm was correct, and we also argued that any database designed using the algorithm will be secure.

Our algorithm did not guarantee that the attributes are not overclassified. This led us to a formal study of the association-based constraints. We provided a way of treating them as a form of data dependency. Subsequently, we introduced the notion of association dependency and described a set of inference rules that will generate all association dependencies from a given set of dependencies. Finally we described techniques for handling the simple and logical constraints during database design.

The next step will be to incorporate the techniques we have designed in order to develop a tool for multilevel database design. In addition, our work on association dependencies should be extended further so that a theory of multilevel relational databases is developed.

ACKNOWLEDGEMENT

We gratefully acknowledge the Department of the Navy (SPAWAR) for sponsoring our work on the Inference Problem under contract F19628-89-C-0001. We thank Dr. Jonathan Millen for his comments on this paper. We also thank Deborah Conte and Reid Gerhart for their comments on an earlier version of this paper.

REFERENCES

[DENN86] Denning, D. E., et al., April 1986, "Views as a Mechanism for Classification in Multilevel Secure Database Management Systems, " Proceedings of the IEEE Symposium on Security and Privacy, Oakland, CA.

[DWYE87] Dwyer, P., G. Jelatis, B. Thuraisingham, June 1987, "Multilevel Security in Database Management Systems," *Computers and Security,* Volume 6, No. 3, pp. 252-260.

[FORD90] Ford, W., B. Thuraisingham, and J. Keefe, J., August 1990, "*Design and Implementation of a Database Inference Controller,*" MTR 10963, Volume 1 (Also accepted to be published in Data and Knowledge Engineering Journal).

[COLL90] Collins, M., October 1990, "*Design and Implementation of a Secure Update Processor,* MTR10977 (Also accepted to be presented at the 7th Computer Security Applications Conference).

[GALL78] Gallaire, H., and J. Minker, 1978, *Logic and Databases,* Plenum Press.

[HINK88] Hinke, T., April 1988, "Inference Aggregation Detection in Database Management Systems," Proceedings of the IEEE Symposium on Security and Privacy, Oakland, CA.

[JAJO90] Jajodia, S., and Sandhu, S., May 1990, "Polyinstantiation Integrity in Multilevel Relations," Proceedings of the IEEE Symposium on Security and Privacy, Oakland, CA.

[KEEF89] Keefe, T., B. Thuraisingham, and W. Tsai, March 1989, "Secure Query Processing Strategies," *IEEE Computer,* Volume 22, No. 3, pp. 63-70.

[LLOY87] Lloyd, J., 1987, "*Foundations of Logic Programming,*" Springer Verlag, Heidelberg, Germany.

[LUNT89] Lunt, T., May 1989, "Inference and Aggregation, Facts and Fallacies," Proceedings of the IEEE Symposium on Security and Privacy, Oakland, CA.

[MAIE83] Maier, D., 1983, "Theory of Relational Databases," *Computer Science Press*, Rockville, MD.

[MINK88] Minker, J., 1988, "*Foundations of Deductive Databases*," Morgan Kaufman, CA.

[MORG87] Morgenstern, M., April 1987, "Security and Inference in Multilevel Database and Knowledge Base Systems," Proceedings of the ACM SIGMOD Conference, San Francisco, CA.

[ONUE87] Onuegbe, E., December 1987, "Issues on Polyinstantiation in a Multilevel Relational Data Model," Technical Note, Honeywell Inc.

[SMIT90] Smith, G., May 1990,"Modelling Security-Relevant Data Semantics," Proceedings of the 1990 IEEE Symposium on Security and Privacy, Oakland, CA.

[STAC90] Stachour, P., and B. Thuraisingham, June 1990, "Design of LDV - a Multilevel Secure Relational Database Management System," *IEEE Transactions on Knowledge and Data Engineering*, Volume 2, No. 2.

[THUR87] Thuraisingham, B., December 1987, "Security Checking in Relational Database Management Systems Augmented with Inference Engines," *Computers and Security*, Volume 6, No. 6.

[THUR89] Thuraisingham, B., December 1989, "Secure Query Processing in Intelligent Database Management Systems," Proceedings of the 5th Computer Security Applications Conference, Tucson, AZ.

[THUR90a] Thuraisingham, B., March 1990, "Towards the Design of a Secure Data/Knowledge Base Management System," *Data and Knowledge Engineering Journal*, Volume 5. No. 1.

[THUR90b] Thuraisingham, B., July 1990, *Handling Association-based Constraints for Multilevel Databases*, WP28904, The MITRE Corporation, Bedford, MA, (not currently in public domain).

A NOTE ON SECURITY CONSTRAINT PROCESSING IN A MULTILEVEL SECURE DATABASE MANAGEMENT SYSTEM

Bhavani Thuraisingham, William Ford, and Marie Collins

The MITRE Corporation, Burlington Road, Bedford, MA 01730

Abstract

This paper provides an overview of security constraints and describes an integrated approach to security constraint processing

1 INTRODUCTION

Security constraints are rules which assign security levels to the data. They can be used either as integrity rules, derivation rules, or as schema rules (such as data dependencies). If they are used as integrity rules, then they must be satisfied by the data in the multilevel database. If they are used as derivation rules, they are applied to the data during query processing. If they are used as data dependencies, they must be satisfied by the schema of the multilevel database.

We have defined various types of security constraints. They include the following:

(i) Constraints that classify a database, relation, or an attribute. These constraints are called simple constraints.

(ii) Constraints that classify any part of the database depending on the value of some data. These constraints are called content-based constraints.

(iii) Constraints that classify any part of the database depending on the occurrence of some real-world event. These constraints are called event-based constraints.

(iv) Constraints that classify associations between data (such as tuples, attributes, elements, etc.). These constraints are called association-based constraints.

(v) Constraints that classify any part of the database depending on the information that has been previously released. These constraints are called release-based constraints. We have identified two types of release-based constraints. One is the general release constraint which classifies an entire attribute depending on whether any value of another attribute has been released. The other is the individual release constraint which classifies a value of an attribute depending on whether a value of another attribute has been released.

(vi) Constraints that classify collections of data. These constraints are called aggregate constraints.

(vii) Constraints that classify any part of the database depending on the security level of some data. These constraints are called level-based constraints.

(viii) Constraints which specify implications. These are called logical constraints.

(ix) Constraint which classify constraints and metadata. These are called meta-constraints.

In our approach, some constraints are processed during the query operation, some during the update operation, and some during database design. Our integrated approach is described in section 2.

2 INTEGRATED APPROACH

As stated in section 1, security constraints enforce a classification policy. Therefore, it is essential that constraints are manipulated only by an authorized individual. In our approach, constraints are maintained by the SSO. That is, constraints are protected from ordinary users. We assume that constraints themselves could be classified at different security levels. However they are stored at system-high. The constraint manager, which is trusted,[1] will ensure that a user can read the constraints classified only at or below his level.

Our approach to security constraint processing is to handle certain constraints during query processing, certain constraints during database updates, and certain constraints during database design. The first step was to decide whether a particular constraint should be processed during the query, update, or database design operation. After some consideration, we felt that it was important for the query constraint processor to have the ability to handle all of the security constraints. Our thesis is that inferences can be most effectively handled, and thus prevented, during query processing. This is because most users usually build their reservoir of knowledge from responses that they receive by querying the database. It is from this reservoir of knowledge that they infer unauthorized information. Moreover, no matter how securely the database has been designed, users could eventually violate security by inference because they are continuously updating their reservoir of knowledge as the world evolves. It is not feasible to have to redesign the database simultaneously.

The next step was to decide which of the security constraints should be handled during database updates. After some consideration, we felt that except for some types of constraints, such as the release and aggregate constraints, the others could be processed during the update operation. However, techniques for handling constraints during database updates could be quite complex, as the security levels of the data already in the database could be affected by the data being updated. Therefore, initially our algorithms handle only the simple and content-based constraints during database updates.

The constraints that seemed appropriate to handle during the database design operation were those that classified an attribute or collections of attributes taken together. These include the simple and association-based constraints. For example, association-based constraints classify the relationships between attributes. Such relationships are specified by the schema and therefore such constraints could be handled when the schema is specified. Since a logical constraint is a rule which specifies the implication of an attribute from a set of attributes, it can also be handled during database design.

1 We assume that a trusted process is a process whose functions are security critical (that is, enforces part of the security policy). Therefore it must be ensured that it operates correctly. The techniques used to ensure its correctness depend on the level of assurance that is expected of the system.

Note that some constraints can be handled in more than one way. For example, we have the facility to handle the content-based constraints during query processing as well as during database updates. However, it may not be necessary to handle a constraint in more than one place. For example, if the content-based constraints are satisfied during the database update operation, then it may not be necessary to examine them during query processing also. Furthermore, the query operation is performed more frequently than the update operation. Therefore, it is important to minimize the operations performed by the query constraint processor as much as possible to improve performance. However, there must be a way to handle all of the constraints during query processing. This is because if the real-world is dynamic, then the database data may not satisfy all of the constraints that are enforced as integrity rules, or the schema may not satisfy the constraints that are processed during database design. This means that there must be a trigger which informs the query constraint processor that the multilevel database or the schema is not consistent with the real-world; in which case the query constraint processor can examine the additional constraints.

Below, we briefly illustrate the architectures for processing constraints during the query, update, and database design operations. The architecture for query processing is shown in figure 1. This architecture can be regarded as a loose coupling between a multilevel relational database management system and a deductive manager. The deductive manager is what we have called the query constraint processor. It has to operate on-line.[2]

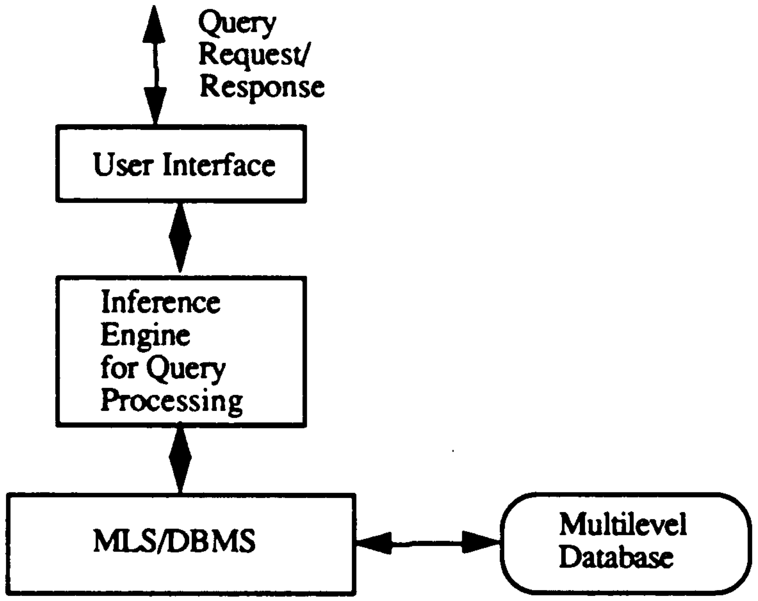


Figure 1. Query Constraint Processor

2 We use the terms Inference Engine and Deductive Manager to mean the same thing. An Inference Engine makes deductions from data and knowledge. For our application it processes security constraints, makes logical deductions, and determines whether there is a potential security violation via inference.

The architecture for update processing is shown in figure 2. This architecture can be regarded as a loose coupling between a multilevel relational database management system and a deductive manager. The deductive manager is what we have called the update constraint processor. It could be used on-line where the security levels of the data are determined during database inserts and updates, or it could be used off-line as a tool that ensures that data entered via bulk data loads and bulk data updates is accurately labelled. If the tool is used off-line, however, it may be difficult to recompute the levels of the data already in the database if these levels are affected by the new data that is being inserted.

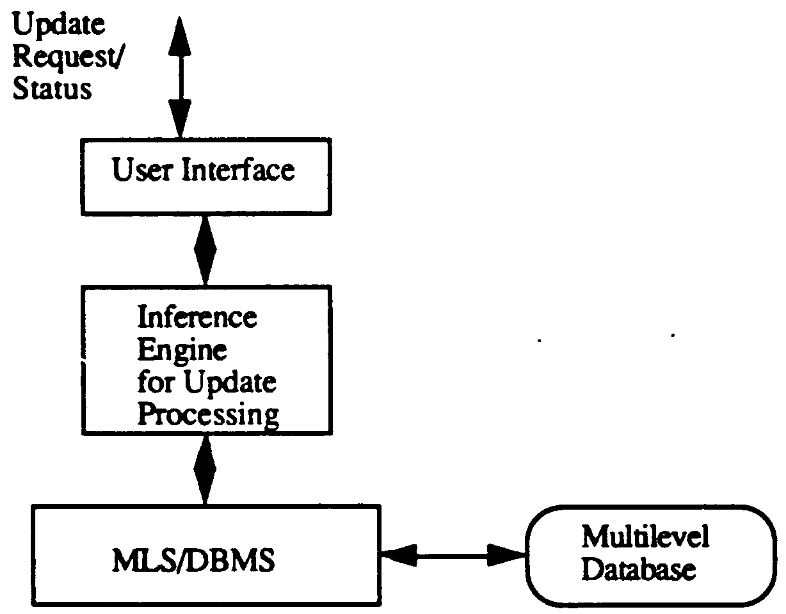


Figure 2. Update Constraint Processor

The tool which handles security constraints during database design, illustrated in figure 3, can be used by the Systems Security Officer to design the schema. The input to the tool is the set of security constraints that should be handled during database design and the schema. The output of the tool is the modified schema and the constraints. We envisage a tool to be operated off-line.

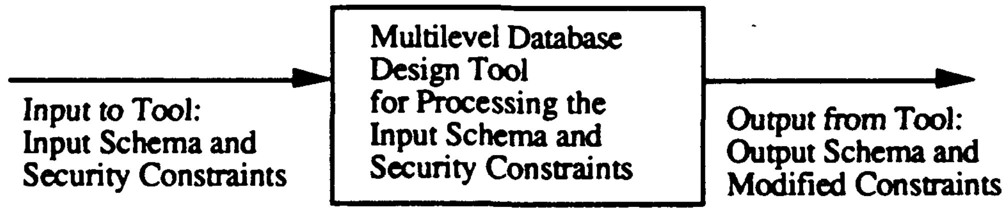


Figure 3. Multilevel Database Design Tool

There are essentially two tasks involved in constraint handling. They are constraint generation and constraint enforcement. While our main focus has been on constraint enforcement (see for example [FORD90, COLL90, THUR91]) the relationship between the two tasks is illustrated in figure 4. That is, the constraint generator takes the specification of the multilevel application and outputs the initial schema and the constraints that must be enforced. The database design tool takes this output as its input and designs the database. The constraints and schema produced by the database design tool are used by the update constraint processor and the query constraint processor.

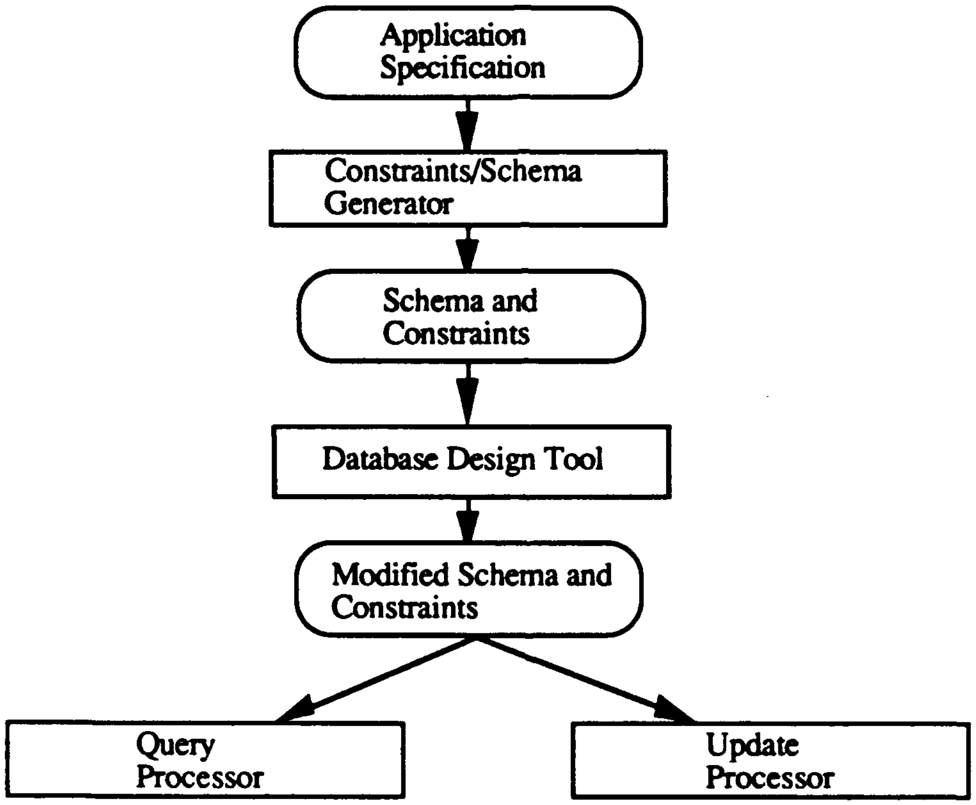

Application Specification
Constraints/Schema Generator
Schema and Constraints
Database Design Tool
Modified Schema and Constraints
Query Processor
Update Processor

Figure 4. Constraint Generation and Enforcement

Although the query constraint processor, update constraint processor, and database design tool are separate modules, they all constitute the solution to constraint processing in multilevel relational databases. That is, these three approaches provide an integrated solution to security constraint processing in a multilevel environment. Figure 5 illustrates the integrated architecture. In this architecture, the constraints and schema which are produced by the constraint generator are processed further by the database design tool. The modified constraints are given to the Constraint Updater in order to update the constraint database. The schema is given to the MLS/DBMS to be stored in the metadatabase. The constraints in the constraint database are used by the query and update constraint processors. We assume that there is a trusted constraint manager process which manages the constraints. In a dynamic environment where the data and the constraints are changing, then the query constraint processor will examine all the relevant constraints and ensure that users do not obtain unauthorized data.[3]

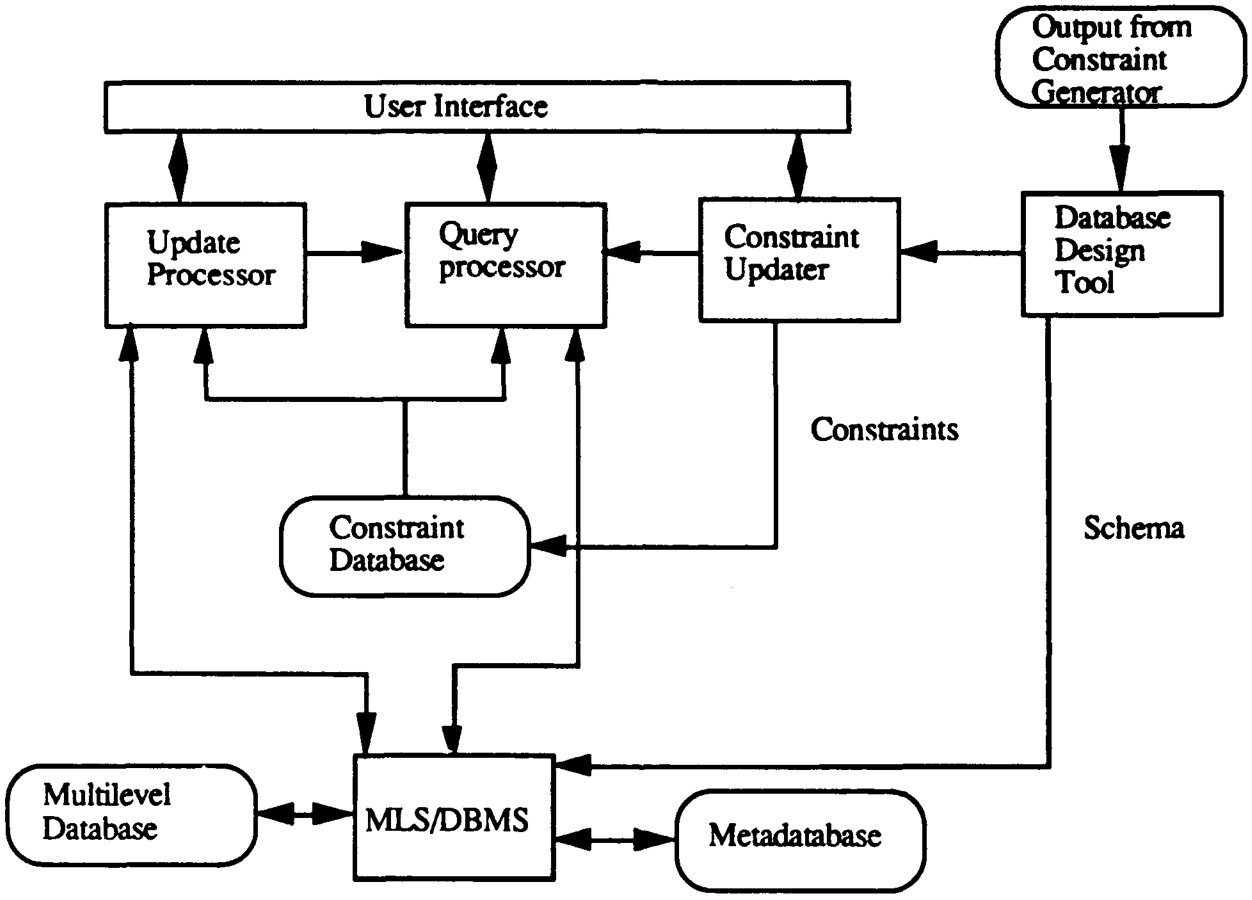


Figure 5. Integrated Architecture

3 Note that if the database is inconsistent, then the security policy for database updates is not satisfied if it is intended that the database must satisfy the constraints at all times. A way to ensure the consistency of the database would be to halt the operation and recompute the levels of the affected data. However, for many applications, especially for time critical ones, this may not be feasible. That is, the system must be operational. This is why we need the query processor to function during times where there could be inconsistencies due to a dynamic environment. The security policy should take this into consideration.

3. SUMMARY AND FUTURE CONSIDERATIONS

In this paper we have defined various types of security constraints and discussed an integrated approach to constraints processing. In particular, some constraints are handled during query processing, some during database updates, and some during database design. The next step will be to implement the integrated architecture described in section 2.

We have focussed mainly on techniques for constraint enforcement. There are also other issues that need to be investigated. Techniques need to be developed to ensure that the constraints are consistent and complete. Techniques also need to be developed to generate an initial set of constraints from the specification of a multilevel database application.

ACKNOWLEDGEMENT

We gratefully acknowledge the U.S. ARMY CECOM for sponsoring our current R&D activities on security constraint processing in a multilevel secure distributed environment under contract DAAB07-91-C-N751. We also acknowledge the Department of the Navy (SPAWAR) for sponsoring our initial R&D activities on security constraint processing under contract F19628-89-C-0001.

REFERENCES

[COLL90] Collins, M., October 1990, *Design and Implementation of a Secure Update Processor*, MTR 10977, The MITRE Corporation, Bedford, MA (a version presented at the 7th Computer Security Applications Conference).

[FORD90] Ford, W., J. O'Keeffe, and B. Thuraisingham, August 1990, *Database Inference Controller: An Overview*, MTR 10963, Volume 1, The MITRE Corporation, Bedford, MA. (a version submitted for publication).

[THUR91] Thuraisingham, B., April 1991, "Handling Security Constraints During Database Design," Presented at the 4th RADC Database Security Workshop, Little Compton, RI.

THE MULTIPOLICY MODEL
A WORKING PAPER

HILARY HOSMER
DATA SECURITY INC.
58 Wilson Road
Bedford, MA 01730

ABSTRACT

The Multipolicy Model permits a multilevel secure (MLS) system to enforce multiple, perhaps contradictory security policies. Data carries policy domain codes to indicate which security policies should be enforced on this data and multiple label segments to supply the attributes needed for each policy. Metapolicies coordinate the enforcement of multiple security policies. Multiple security policies are maintained independently by multiple policy domain administrators.

This paper illustrates that the current TCSEC security policy paradigm can be enlarged to meet the needs of a more interrelated and integrated world. Multiple policies permit more natural modelling of real-world security practices and allow easier sharing of data among users in different security domains. Commercial applications include medical, financial, and investigative systems that cross policy domains. Military applications include communication, command, and control (C3) systems in multinational and multiservice battle theaters.

INTRODUCTION

Integrating security policies on today's multilevel secure (MLS) computer systems is a difficult, sometimes impossible problem[1]. When the security policies themselves cannot be integrated, the systems built to implement these policies cannot be integrated either.

Sometimes the only way to solve impossible problems is to transcend them. For example, when astronomers found the mathematics of an earth-centered planetary system impossibly complicated, Copernicus developed a new model with the sun at the center. His paradigm simplified planetary astronomy and initiated waves of discovery by others. Thomas Kuhn documents a number of these ground-breaking paradigm shifts in his book, *The Structure of Scientific Revolutions*.[2] Hoping for similar breakthroughs, computer security founder Dr. Willis Ware has called for a new MLS paradigm which will make networking and integration of MLS systems easier.

This paper outlines a model based upon multiple policies which may become a new security paradigm. It describes the current paradigm, identifies some of its shortcomings, and summarizes some of the requirements for an alternate paradigm. It proposes a new model, and describes its components and workings. Finally, the paper reviews possible implementation strategies, applications and benefits, and raises critical issues.

The author presented an earlier version of this paper at the National Institute of Standards and Technology (NIST) on April 10, 1991[3] and convinced the audience that the GOSIP standard label should leave room for multiple policies. This paper expands on the earlier work, exploring the components of a Multipolicy Model in more detail.

This paper is being presented at the Fourth RADC Multilevel Database Security Workshop for two reasons. First, the Multipolicy Model may help solve some of the 'impossible' MLS DBMS problems, such as integrating MLS application policies with MLS DBMS and MLS operating system policies. Secondly, the work on the Multipolicy Model is just beginning, and has received no funding to date. Hopefully, the attention the model receives at the RADC workshop will encourage the financial support necessary to develop a substantial and quantitative model.

[1] Hosmer, Hilary, "Integrating Security Policies", Proceedings of the Third MLS DBMS Workshop, Castile, NY. June 1990, MITRE Technical Paper MTP 385.

[2] Kuhn, Thomas, The Structure of Scientific Revolutions, 2nd Edition, University of Chicago Press, Chicago, 1970.

[3] Hosmer, Hilary H. "The Multipolicy Machine: A New Paradigm for Multilevel Security Systems" (Position Paper), Standard Security Label For GOSIP, An Invitational Workshop, NISTIR 4614, June 1991

THE CURRENT PARADIGM

The TCSEC

The Trusted Computer System Evaluation Criteria (TCSEC or Orange Book) embodies the United States' security paradigm.[4] It assumes a single "system security policy" which can be divided into major subpolicies such as Confidentiality, Integrity, and Assurance of Service. The subpolicies are further subdivided. For example, Confidentiality can be divided into Access Control and Non-Access Control policies, and Access Control policies can be subdivided into Mandatory Access Control (MAC) and Discretionary Access Control (DAC). However, the TCSEC paradigm assumes that all these subpolicies cohere together to represent one overall system security policy. The single overall policy drives the choice of security mechanisms and is the foundation of most assurance efforts.

The single-policy paradigm works well with stand-alone systems but causes problems when systems must be networked or combined and security policy integration is required. For example, when MLS products with slight variations in policies, such as Operating System (OS), Database Management System (DBMS), and user applications, must work together, there may be policy integration difficulties as well as other interoperability issues.[5] The same holds true when systems enforcing different policies, such as U.S.A. Department of Defense (DOD), North Atlantic Treaty Organization (NATO), European Community (EC), and France, must interact and share classified data. Interoperability often requires compromises.

The TNI

The Trusted Network Interpretation (TNI) of the TCSEC enlarges the single-policy paradigm so that multiple policies may coexist on networks. It permits each node on a network to have its own nodal security policy, but stipulates that the network as a whole must have an overall global network security policy which is used as the basis for evaluating the security of the network[6].

The TDI

The Trusted Database Interpretation (TDI) focuses on evaluating Trusted Computing Base (TCB) subsets. Each subset can enforce a different security policy, such as MAC

[4] Department of Defense, Department of Defense Trusted Computer System Evaluation Criteria, DOD 5200.28-STD, December 1985.

[5] Hosmer, "Integrating Security Policies", op cit.

[6] National Computer Security Center, Trusted Network Interpretation of the Trusted Computer System Evaluation Criteria, 31 July 1987.

or DAC. The TDI assumes, however, that these subpolicies cohere into a single consistent overall security policy[7].

The ITSEC

The draft International Technology Security Evaluation Criteria (ITSEC)[8] permits a user to specify a security policy, select a system meeting his needs, then request a certified evaluation center to do an evaluation to provide the necessary assurance that the selected system is able, in fact, to carry out the user's security policy. There is no restriction on what functionality could be in the user's policy. The policy could include integrity, availability, non-repudiation, and cryptography, for example. The ITSEC follows the TCSEC lead in requiring users to integrate multiple separate policies into a single coherent system security policy.

Problems With The Current Paradigm

The paradigm of a single security policy with multiple coherent subpolicies has some major shortcomings which are becoming apparent now that multilevel secure systems are being fielded.[9]

> *It's inflexible.* If a user wants to modify built-in aspects of the system security policy, the whole system must be reevaluated.
>
> *Exchanging sensitive data with systems with other security policies is difficult or impossible in real-time.* Guards are needed at all interfaces, and mapping rarely can translate security levels from one policy to the other without upgrading.
>
> *It's unrealistic.* The real world has multiple coexistent security policies. A computer security officer creating an automated security policy[10] must often integrate diverse and contradictory security policies together into a single coherent policy to meet TCSEC criteria. Canada's experience trying to integrate the national privacy policy with the national

[7] National Computer Security Center, Trusted Database Interpretation of the Trusted Computer System Evaluation Criteria, April 1991

[8] Information Technology Security Evaluation Criteria, draft of 2 May 1991 Version 1. These 'harmonised' criteria are based largely on the German "Blue Book" plus other European criteria and are a superset of the TCSEC. Copies are available from Dr. James Burrows at NIST in Gaithersburg MD, or Dr. Hartwig Kreutz at the German Information Security Agency in Bonn Germany.

[9] For context, the next two sections and the later section on implementation are taken from "The Multipolicy Machine: A New Paradigm For Multilevel Secure Systems", op cit.

[10] Sterne, Daniel, "On the Buzzword Security Policy", Proceedings of the 1991 IEEE Computer Security Symposium on Research in Security and Privacy, May 1991, Oakland, California.

disclosure policy into a single policy lattice illustrates the real difficulties users face [11].

Performance is poor. Adding security to existing systems seriously slows down throughput.

The current paradigm must be enlarged to meet the needs of a more interrelated and integrated world. With a few significant enhancements, the single-policy paradigm can be extended into a more flexible, more interoperative, better-performing multipolicy paradigm.

REQUIREMENTS FOR A NEW PARADIGM

What must a larger and more inclusive paradigm do? It should:

Handle bottom-up system construction. The end-user, supposedly the originator of the system security policy, can change only the policy lattice values and system parameters without reevaluation. We need a paradigm which permits the end-user to establish much more of his own security policy without requiring a reevaluation of the whole system.

Separate the policy from the enforcement mechanism. Because of the single-policy paradigm, current trusted systems implement the system security policy as an integral part of the system, making it impossible to separate the policy from the mechanisms which implement that policy.[12] A more flexible paradigm would separate policy from mechanism so that mechanisms can enforce more than one policy, and policies can be tailored.[13]

Ease integration of trusted system components. Under the single-policy paradigm, each purchased component, including hardware, operating system, DBMS, and applications packages, must be integrated into a coherent package that can be proven to implement the end-user's security policy. This integration is difficult when diverse vendors' components implement slightly different security policies or slightly different versions of the same security policy. We need

[11]The difficulties are clearly illustrated in D. S. Crawford's account of of efforts trying to integrate the Canadian Privacy Policy with the Canadian Disclosure policy into a single coherent policy. See Crawford, D.S. "Modelling Security Policy and Labelling Unclassified but Sensitive Information - A Canadian Perspective", Proceedings of Standard Security Label for GOSIP An Invitational Workshop, NISTIR 4614, June 1991 .

[12] An example of this appears in Jackson Wilson's "A Security Policy for an A1 DBMS (a Trusted Subject)", Proceedings of the 1989 IEEE Computer Security Symposium on Security and Privacy, May 1-3, Oakland, California.

[13] An example of separation appears in Grenier, Guy-L, Richard Holt, Mark Funkenhauser, "Policy VS Mechanism in the Secure Tunis Operating System", Proceedings of the 1989 IEEE Computer Society Symposium on Security and Privacy, May 1-3, 1989, Oakland, California.

both standards and a model which accommodates policy variations.

Ease sharing data with other policy systems. The single-policy paradigm founders on the pressing need to share data with others who have different security policies. In multinational conflicts like the Persian Gulf, US DOD users need to share classified data with allied computers that implement different national or international security policies.

Enforce the originator's security policy. Current strategies for sharing data across security policy boundaries (Guards, Man-in-the-loop) frequently must upgrade or downgrade data, thus losing the original classification. The assessment time required for human downgrading makes it difficult to share data in real-time in a fast-moving multinational battlefield situation. Even if the multinational situation is one of cooperation rather than conflict (for example, divisions of a multinational corporation, or international electronic funds transfer), we would like to be able to enforce the originator's security policy while sharing data among computer systems.

Permit contradictory policies to operate in parallel. To use a medical example, different states have passed different laws about releasing AIDS data. If an AIDS patient from Connecticut is in a New York hospital, which laws should apply to the release of data, New York's or Connecticut's or both? Similarly, different hospitals have very different policies about release of data to insurance companies. Some, for convenience, give a patient's entire medical record when an insurance company makes a request for information. Others provide only the data that the insurer has requested and needs to know. If a patient of a doctor at the need-to-know hospital must go for surgery to the other, will the second hospital protect the entire medical record in accordance with the first hospital's policy or its own? Release of sensitive medical data benefits health insurance companies who review medical records for evidence of risk of AIDS and terminate insurance for high risk clients.

The current TCSEC/TNI/TDI policy paradigm may preclude systems such as a national AIDS databank which enforces many different Mandatory Access Control (MAC) policies (one for each state, plus one for the nation). It makes it difficult to build the European Community health system where the varying disclosure laws of 12 different countries must be implemented and maintained. A new paradigm which permits contradictory organizational policies to operate in parallel is needed.

Improve the performance of trusted systems. Adding security to a system usually degrades its performance significantly, largely because of access control checks and auditing.

Other. The list above is not exhaustive. As more multilevel systems are implemented, we will become aware of more difficulties and requirements.

Solving these problems is essential to widespread user acceptance of MLS systems.

THE MULTIPOLICY MODEL

The Opportunity

The Multipolicy Model will solve significant portions of these long-standing problems. First, it provides a vehicle for users to add their own security policies to a system without disrupting or invalidating existing evaluated policies. Secondly, it eases integration problems by preserving the original classification of data when data is passed across policy boundaries. Thirdly, it permits one machine to enforce a variety of parallel security policies which are not necessarily consistent with one another. Fourthly, it may improve trusted system performance by being implemented in high-speed parallel processing architecture.

The Components

Like any security model, the Multipolicy Model has Subjects, Objects, definitions of allowable operations, and a Reference Monitor to mediate subjects access to objects in accordance with the rules. Like any security model with mandatory access control (MAC), Labels are associated with Subjects and Objects and used to determine the attributes of subjects and objects for mediation purposes. As in any security model, security policy data is stored in a policy Lattice, which can be initialized and modified by the System Security Officer.

However, in order to support multiple policies, the Multipolicy Model requires several components which do not appear in current security models. There are multiples of each of the following:

(1) Security Policy Domains[14],

[14] ECMA "Security in Open Systems: A Security Framework", ECMA TR/46, July 1988.

(2) Security Policy Domain Codes[15]
(3) Labels or label segments[16],
(4) Security policies, including
  (a) Allowable operations
  (b) Policy lattices
  (c) Separate user policies
(5) Metapolicies[17]
(6) Security Policy Enforcers.

Each of these components is described in detail below.

Security Policy Domains

Today there are many different authorities responsible for maintaining various levels of security policy: ISO, ECMA, DoD, AF, AFB, to name a few. The general approach to creating a computer system security policy is to integrate all these policies and represent the combined policy on the computer. This technique makes it very difficult to change the system security policy when any of the organizational security policies changes.

The Multipolicy Model permits multiple distinct security policy domains, administered by different organizational entities each with complete policy autonomy in its domain, to be modeled in a computer system.

Domains may be autonomous, each with its own subjects, objects, policies, and reference monitor. A multipolicy model implemented on the Amdahl secure computer[18] with its seven isolated domains might work this way. Domains may cross each other, so that subjects or objects may belong to more than one domain and fall under more than one policy. (The rules for controlling the interactions of multiple policies is described later under 'Metapolicies').

Security policies change rarely. When required, however, Domain Administrators can implement changes in policy in their domains in a variety of ways. For example, if they are located remotely, they could "securely download" new policy lattices and/or code, send revised firmware chips for installation by each System Security Officer (SSO), or simply work through the SSO directly.

In a multipolicy situation, an object may fall under one domain or another or even multiple domains. **Security policy domain codes** are associated with security labels or label segments and indicate which policy domain applies to this

[15] NIST "Proposed Standard Label for GOSIP", RIC (Register Index Code)

[16] Hosmer, H.H. Presentation at NIST, Invitational GOSIP Standard workshop, April 10, 1991.

[17] Hosmer, "The Multipolicy Machine..." op cit .

[18] Amdahl product literature, courtesy of Chief Architect Jon Graff.

label. For example, an object which falls under both Canadian and US security policies would need two policy domain codes, one for the US policy and one for the Canadian policy. Whenever access control decisions are made, these policy domain codes are checked first so that the proper policy enforcers and policy lattices can be invoked.

Security Policy Domain Codes

Every object must have one or more codes indicating which security policy or policies apply to the object. These codes are associated with the security label and indicate to the security policy enforcer how to interpret the security attributes in the label.

Example Format:

OBJECT / SECURITY ATTRIBUTES / POLICY CODE

Examples:

Patient / John Jones / 10010001001 / Privacy-NY

The string of bits representing patient Jones' release permissions will be interpreted in accordance with the New York privacy format and the New York privacy policy.

This policy domain code format resembles the proposed European Computer Manufacturers Association (ECMA) security domain codes on security labels which indicate under which label convention the label is formatted, eg. International Standards Organization (ISO)[19]. These policy domain codes could easily be implemented using the proposed NIST GOSIP Register Index Code (RIC). The RIC points to the semantic definition of the label as registered in the Security Objects Register of a standards organization, such as the proposed NIST Computer Security Object Register[20].

Multiple labels or label segments

Multiple labels or label segments describe the security attributes for the object. There will be one label segment for each policy that applies to the object. On a small scale, this is already a common practice. For example, DEC's secure operating system SEVMS provides a security label which may be divided into a Sensitivity label segment and an Integrity label segment.

[19] ECMA TR/46, Security in Open Systems, A Security Framework, European Computer Manufacturers Association, Geneva, Switzerland, July 1988.

[20] Standard Security Label for GOSIP, An Invitational Workshop, op. cit.

If an object comes under control of more than one policy, a set of attributes is needed for each policy. These could be handled as multiple labels, a strategy used by some network protocols. However, in order to preserve the intuitive notion that each object has one security label, in this paradigm each set of attributes will be implemented as a label segment accompanied by a policy domain code. This was proposed by the author for the NIST GOSIP standard label, and accepted for the last internal draft standard.

Example Format:

OBJECT / LABEL / POLICY / LABEL / POLICY / etc.
SEG / CODE SEG

Examples:

Patient/ Jon Jones / 01001001010 / Privacy-MASS/ 01001000001/ Privacy-CONN / 010010101111 /Privacy-NY

Patient Jones, who lives in Connecticut, is hospitalized in New York and then sent for consultation to a teaching hospital in Massachusetts. The privacy policies for all three states apply to him and his hospital record has three sets of privacy attributes.

To avoid representing attributes in binary, the attributes will be summarized by type in subsequent examples.

Example:

Military Target data / Integrity Attributes / BIBA /Sensitivity Attributes / US-DOD

Under the single-policy model, a set of security policies is normally integrated into one single policy. For example, at an Air Force base, the DOD security policy, the Air Force security policy, and the Base security policies are integrated and implemented as a single automated security policy. This is not too difficult with hierarchical policies where subordinate policies automatically inherit the policies of their superior.

However, under the Multipolicy Model, each of the hierarchical policies - the DOD policy, the Air Force policy, and the Air Force Base (AFB) policy - would be separate policies, and a policy domain code would be required for each. This gives the AFB security administrator the flexibility to change local base policy while leaving national DOD and AFB policies untouched.

Example: **Object / Security Attributes / DOD /Security Attributes/ AF / Security Attributes / AFB**

Note that metapolicy rules in such hierarchical systems always give precedence in case of conflict to the higher policy.

Metapolicies

Metapolicies are policies about policies. They coordinate the interaction between policies, specifying order, priority, and conflict-resolution strategies.

A metapolicy includes:

1. Who can set policy, who can change policy, and what the procedures are for changing policies.

2. Rules about developing, verifying, and protecting security policies.

3. Rules about the interaction of multiple security policies, especially where they conflict. These rules will specify:

   A. Multiple Policy Relationships. The ways that the multiple policies relate to each other could be hierarchical, serial, parallel, overlaid, or circular.

   B. Policy Interactions. The ways that multiple policies interact with each other could include inheritance, mutual exclusivity, mutual dependence, and other.

   C. Precedence rules. Where should the enforcer begin enforcing multiple policies? In what order should they be enforced?

   D. Documentation. Metapolicies should be included in both formal and informal models.

   E. Transfers of control among policies and after all policies are enforced.

Security Policies

As in a single policy machine, security policies consist of: a) definitions of subjects and objects; b) definitions of allowable operations; and c) the rules of the policy, including a policy lattice for ordering sensitivity levels, integrity levels, compartments, etc. As in the single-policy machine, each policy is separate from the others and tamperproof. However, each computer system will be able to enforce more than one policy. If appropriate, a single node could enforce every policy implemented in the network.

One of the frustrations experienced by users is their limited ability to modify the security policies which are built into the system. The capacity to absorb multiple user policies (representing multiple nations, multiple divisions, or several kinds of integrity policies) without reevaluating the whole system is an integral part of the Multipolicy Model. One implementation method would be for the SSO to enter user policies via trusted software. Each policy is stored in an isolated area, and its interactions with other evaluated policies are controlled by the appropriate metapolicies.

Security Policy Enforcers

Security policy enforcers implement the rules of a policy on the subjects and objects. Access control policy enforcers and other non-access control policy enforcers will be needed. Each enforcer is trusted to protect and enforce the policies in its domain correctly and must be tamperproof.

Multiple policies may be enforced in several different ways. A single reference monitor may access multiple independent policy lattices, as occurs now with Secrecy and Integrity policies. Multiple reference monitors may enforce multiple different policies, or multiple versions of the same policy, or multiple subsets of the same policy.

CROSSING POLICY BOUNDARIES

A definition of the allowable operations is an integral part of a security policy. This section defines some of the critical operations and permitted interactions of the Multipolicy Model.

When an object is exported from one machine to another, what happens in the multipolicy environment?

It is possible for an object to go from one machine to another without leaving its original policy domain. The receiving machine implements more than one policy and may well implement the policy indicated in the label associated with the object. If the receiving machine does not enforce the object's policy or policies, it must either pass the object on to another machine which enforces the appropriate security policy, or hold on to the object without permitting any access. Which choice is made could depend upon instructions which accompany the object, or on the metapolicy for the receiving computer.

IMPLEMENTATION OPTIONS

There are several reasonable approaches to the implementation of a multipolicy model.

1. Multiple sets of rule-based policies;
2. Multiple co-processors;
3. Distributed policies;
4. Parallel processors;
5. Multidomain machines;
6. Redundant fault-tolerant policies.

Each is described below.

Rule-based

Several researchers, including Page, Heaney, Adkins, and Dolsen of Planning Research Corporation[21] and Abrams, LaPadula, Eggers and Olsen of MITRE[22], have been exploring Rule-Based access control policies. The Rule-Based concept permits security policies to be implemented as sets of rules and has been formally modelled by Dr. La Padula[23].

The Multipolicy Model could be implemented with multiple sets of rule-based access control policies implemented in software or firmware. There would be separate sets of rules for users, with a trusted user interface so that user policies could be changed by user authorities independently or with only loose communication with the System Security Officer.

[21] **Page, John, Jody Heaney, Marc Adkins, Gary Dolsen, "Evaluation of Security Model Rule Bases", Proceedings of the 12th National Computer Security Conference, Baltimore, Maryland, 1989.**

[22] **Abrams, Marshall, Leonard LaPadula, Kenneth Eggers, Ingrid Olson, "A Generalized Framework for Access Control: An Informal Description", Proceedings of the 13th National Computer Security Conference, Washington, D.C., October 1990.**

[23] **La Padula, Leonard, "Formal Modeling in a Generalized Framework for Access Control ", Proceedings of the Computer Security Foundations Workshop III, Franconia N.H., June 1990.**

Mulciple Co-processors

A second approach is to use multiple coprocessors, such as LOCK (Logical Coprocessing Kernel), to implement multiple policies. Although LOCK has an integral built-in security policy, it provides for kernel extensions which can handle different policies under control of the sidearm.[24] A multipolicy machine could, in theory, be constructed out of many single-policy coprocessors operating in parallel, improving processing speed.

Distributed System

A third approach is to use a distributed system where each machine implements a local security policy, and data whose sensitivity prevents it from being processed on one machine is forwarded to another. This approach could be used with current trusted equipment, although it would not be very efficient.

For efficiency, each local machine should implement all the local security policies, and data which doesn't come under the local policies would be forwarded to a remote node for policy enforcement. A master policy node which be able to enforce all or most of the policies on the system.

The distributed approach assures that local policies will be applied quickly, while keeping the capability for implementing rare policies.

Parallel Processors

Very large scale integrated circuits (VLSI) make it possible to build trusted systems in hardware. Processors on a chip make it possible for each policy and its enforcer to operate in parallel with other policies and enforcers.

Multidomain Machines

Some secure computers, such as Amdahl's, provide a number of separate domains which could be used to support a finite number of different security policies.

Hybrids

Many combinations of the above techniques would be possible, as illustrated with the second distributed example. Other approaches not mentioned here are possible as well.

[24] Saydjari, O. Sami, Joseph Beckman, Jeffrey Leaman, "LOCK Trek: Navigating Uncharted Space", Proceedings of the 1989 IEEE Computer Society Symposium on Security and Privacy, May 1-3, 1989, Oakland, California

APPLICATIONS

The Multipolicy Model is most useful where there clearly are multiple security policies involved. For example, the Europeans are planning a European Community Health System which would cover the citizens of many countries. Now several nations are trying to develop a single unified security policy for the international health system. Given the different values and temperaments of the nations involved, it might be more fruitful to set up a multipolicy system so that different nations can independently control the security policies for their citizens. This is particularly important in the area of privacy where different values prevail in different communities.

There are many military multipolicy applications. Multinational battle management, multinational command and control centers, logistics involving multiple services, and multinational communications systems are just a few. Often there is one information security policy in peacetime which is loosened when war starts because security is less critical than response time or other factors. This is also a multipolicy situation and might be handled best by defining both policies and shifting from one to the other rather than "loosening" the more restrictive policy.

Commercial applications for multipolicy machines are numerous. There is no single standard security policy, like that of the DOD, in the commercial world, so a secure system, to be marketable, must be able to adapt to multiple policies. Of course, extremely knowledgeable people can with ingenuity adapt the TCSEC to meet a wide range of needs. Dr. Bell's paper[25] on the commonality of security policies illustrates this clearly. However, most users are neither so knowledgeable nor so ingenious, and the Multipolicy Model would facilitate expression of their diverse and unanticipated security policies.

Multinational banks, multinational corporations, international non-profit activities such as the Red Cross and CARE, merged corporations with multiple corporate cultures, colleges and companies which cross state borders, international telecommunications systems, all are candidates for multipolicy systems.

[25] Bell, D. Elliott, "Putting Policy Commonalities to Work", Proceedings 14th National Computer Security Conference, Washington, D.C. October 1990.

ISSUES

In spite of the wide range of applications for the Multipolicy Model, there are several important questions to ask about it. Here are some anticipated questions, and possible answers to the concerns expressed.

Q. Is a Multipolicy mModel really necessary? Can't we do it all with the TCSEC?

A. The Multipolicy Model, when developed, will be more elegant, more flexible, and easier to use than the TCSEC. The Multipolicy Model naturally models the world outside the DOD, and will become more important as integrity, privacy and ethical concerns creep to the fore.

Q. Will the National Computer Security Center (NCSC) accept the Multipolicy Model?

A. If the details are sufficiently worked out to prove that it is secure, the NCSC would welcome a new paradigm, especially if it does not invalidate the excellent work in security accomplished to date. The Electronic Systems Division of the U.S. Air Force plans to fund a feasibility study of the Multipolicy Machine via a Small Business Innovation Research (SBIR) Phase I grant to Data Security Inc. Starting in Sept. 1991, we start exploring these and other issues.

Q. Several national and international agencies (ECMA and ISO, for example) are working on sensitivity label standards to make information interchange easier between MLS systems. Can the Multipolicy Model incorporate these evolving standards?

A. Yes. The Multipolicy Model incorporates European standards and is helping to shape US standards. These standards are included in the discussion on labels and domain codes.

Q. Can the Multipolicy Model be used on a machine which is simple to manufacture, evaluate, and accredit? Can commercial off-the-shelf components be used?

A. Hopefully, the answer to both questions is yes. Several vendors who have seen the model are claiming that their machine will implement such a model. SCTC and Amdahl are among them.

Q. How much more complicated will it be to evaluate multiple instead of single policy machines?

A. Although initially more difficult, it will eventually be easier to evaluate multiple policy machines than single policy machines because the policy will be separate from the mechanisms. Now, policy and mechanisms are integrated and must be evaluated together. When rule-based or other machines which separate policy from mechanism are accepted, it will be sufficient for the vendor to prove to the evaluators that their mechanisms implement any of a set of security policies. Proving that the particular policy of a particular installation is valid and supported by the mechanism is left to the certification and accreditation process.

Q. Where will the Multipolicy Model be the most useful?

A. The Multipolicy Model will be most useful in networks, which require higher levels of either computer or physical security. The Multipolicy Model will probably be built first in Europe where the need to cross security domain boundaries is well-established and understood.

CONCLUSIONS

The Multipolicy Model is a security model which could be successfully implemented in many ways. It will provide greater flexibility for users who need to add their own security policy specifics to the security policy of an existing system. It will make it easier to transfer data to systems in other security policy domains. It will let users model complex real world security policies more easily and permit contradictory policies to operate in parallel. Parallel processing may permit an improvement in trusted system performance, as well.

The Multipolicy Model is now just a concept with potential. Much more work needs to be done to make it a reality.

REFERENCES

Abrams, Marshall, Leonard LaPadula, Kenneth Eggers, Ingrid Olson, "A Generalized Framework for Access Control: An Informal Description", *Proceedings of the 13th National Computer Security Conference*, Washington, D.C., October 1990.

Bell, D. Elliott, "Putting Policy Commonalities to Work", *Proceedings 14th National Computer Security Conference*, Washington D.C., Oct. 1991.

Bell, D. Elliott, "Lattices, Policies, and Implementations", *Proceedings 13th National Computer Security Conference*, Washington D.C. October 1990.

Bell, D. Elliott and L.J. La Padula, "Secure Computer Systems: Unified Exposition and Multics Interpretation", MTR-2997, The MITRE Corporation, Bedford, MA, July 1975 (ESD-TR-75-306).

Biba, K.J., *Integrity Considerations for Secure Computer Systems*, MTR-3153, Rev. 1, Electronic Systems Division, Air Force Systems Command, United States Air Force, Hanscom Air Force Base, Bedford, MA, April 1977 (ESD-TR-76-372).

Department of Defense Trusted Computer Systems Evaluation Criteria, December 1985, DOD 5200.28 STD.

European Computer Manufacturers Association, *Security in Open Systems, A Security Framework*, ECMA TR/46, July 1988.

Hosmer, Hilary H. "Integrating Security Policies", *Proceedings of the Third RADC Workshop of Multilevel Database Security*, Castile, NY, June 1990.

Hosmer, Hilary H. "The Multipolicy Machine: A New Paradigm For Multilevel Secure Systems", *Proceedings of Standard Security Label for GOSIP, an Invitational Workshop*, April 1991, NISTIR 4614, June 1991.

International Technology Security Evaluation Criteria (ITSEC), German Information Security Agency, Bonn, Germany, draft of 2 May 1990.

Kuhn, Thomas, *The Structure of Scientific Revolution*, University of Press, 1970.

LaPadula, Leonard, "Formal Modeling in a Generalized Framework for Access Control", *Proceedings of the Computer Security Foundations Workshop III*, Franconia, N.H. June 1990.

Security Foundations Workshop III, Franconia, N.H. June 1990.

Matley, Ben G and Thomas A. McDannold, *National Computer Policies,* IEEE, 1987.

National Computer Security Center, *Trusted Network Interpretation of the Trusted Computer System Evaluation Criteria (TNI),* 31 July 1987.

National Computer Security Center, *Trusted Database Interpretation of the Trusted Computer System Evaluation Criteria (TDI),* April 1991

Page, John, Jody Heaney, Marc Adkins, Gary Dolsen, "Evaluation of Security Model Rule Bases", *Proceedings of the 12th National Computer Security Conference,* Baltimore, Maryland, 1989.

Saydjari, O.S., J.M. Beckman, and J.R. Leaman, "LOCK Trak: Navigating Uncharted Space", *Proceedings of the 1989 IEEE Computer Symposium on Security and Privacy,* May 1-3, 1989, Oakland, California.

SECTION 6

DISTRIBUTED DATABASE MANAGEMENT

Rome Laboratory has been sponsoring research in trusted distributed database management for several years. In their paper "Trusted Distributed Database Management Systems R&D - A Progress Report," Bhavani M. Thuraisingham and Harvey H. Rubinovitz describe the components of a trusted distributed DBMS, the results of their protyping effort and their current research activities.

TRUSTED DISTRIBUTED DATABASE MANAGEMENT SYSTEMS R&D - A PROGRESS REPORT

Bhavani Thuraisingham and Harvey Rubinovitz

The MITRE Corporation, Burlington Road, Bedford, MA 01730

ABSTRACT

This paper describes research and implementation results on interconnecting (i) homogeneous trusted database management systems and (ii) heterogeneous trusted database management systems.

1. INTRODUCTION

Information has become a critical resource in many organizations, and sharing and accessing information has become a priority. The rapid development of distributed database management systems (DDBMS) will enable this need to be achieved. Interconnecting the increasing number of databases scattered across several sites is gaining popularity, and tools which enable users of one system to use other systems are being developed in order to reconcile the contrasting requirements of the different DBMSs. Efficient approaches for interconnecting different database systems, as well as administering them, are also being sought [SHET90].

The increasing popularity of distributed database systems should not obscure the need to maintain security of operation. That is, while data access and sharing is necessary, it is also important that the system operate securely in order to overcome any malicious corruption of data as well as prohibit unauthorized access to and use of classified data especially with military applications. For many applications, it is especially important that the DDBMS should allow users, who are cleared to different levels, access to the database consisting of data at a variety of sensitivity levels without compromising security. That is, users should only obtain the information to which they are authorized.

A considerable amount of work has been carried out in providing multilevel user/data handling capability in centralized database management systems, known as trusted database management systems (TDBMS) (see, for example, [AFSB83, GRAU85, DENN87, STAC90]). In contrast, trusted distributed database management systems (TDDBMS) are only recently receiving some attention. As TDBMSs are become commercially available, the secure interoperability of these systems will be necessary.

We have taken an incremental approach to R&D in TDDBMSs.[1] Initially, we investigated policy, design and implementation issues for a TDDBMS which functions in a homogeneous environment. In such an environment it is assumed that all of the TDBMSs operate the same way. In addition to R&D in TDDBMS which functions in a homogeneous environment, at present, we have also investigated security issues for interconnecting heterogeneous TDBMSs. The results of our investigation are described in [COLL90, RUBI90a, RUBI90b, THUR91a, RUBI92]. This paper summarizes the progress that we have made.

2. INTERCONNECTING IDENTICAL TDBMSs

In this section we describe our research and implementation results on interconnecting identical TDBMSs. In particular, we discuss: (i) the definition of an architecture and mandatory security policy, (ii) design, simulation and implementation of secure query processing algorithms, and (iii) design and simulation of secure distributed concurrency control algorithms.

2.1 ARCHITECTURE AND POLICY ISSUES

An architecture for interconnecting multiple identical TDBMSs is shown in figure 1 [THUR90]. This architecture has been derived from the choice architecture for a DDBMS given in [CERI84]. In this architecture,

[1] In general, by a TDDBMS we mean one that functions either in a homogeneous or heterogeneous/autonomous environment. A TDDBMS which functions in a homogeneous environment could be based on the multidatabase approach (in which case it consists of a collection of interconnected identical TDBMSs) or it could provide an integrated global schema to all of its users. A TDDBMS which functions in a heterogeneous/autonomous environment is one which interconnects heterogeneous and possibly autonomous TDBMSs. A TDDBMS could be federated if the component TDBMSs participate in one or more federations. The TDDBMS that we have considered in our investigation is based on the multidatabase approach.

the TDDBMS consists of several nodes that are interconnected by a trusted network. All of the nodes are designed identically. Each node is capable of handling multilevel data. Each node has a Relational TDBMS which manages the local multilevel database. Each node also has a distributed processing component called the Secure Distributed Processor (SDP). The components of the SDP are the Distributed Query Processor (DQP), the Distributed Update Processor (DUP), the Distributed Transaction Manager (DTM), and the Distributed Metadata Manager (DMM). The DQP is responsible for distributed query processing. The DUP is responsible for distributed update processing. The DTM is responsible for distributed transaction management. The DMM manages the global metadata. The global metadata includes information on the schemas which describe the relations in the distributed database, the way the relations are fragmented, and the locations of the fragments. SDP may be implemented as a set of processes separate from the local TDBMS. Two DQPs (or DUPs, DTMs, DMMs) at different nodes communicate in accordance with the security policy enforced.

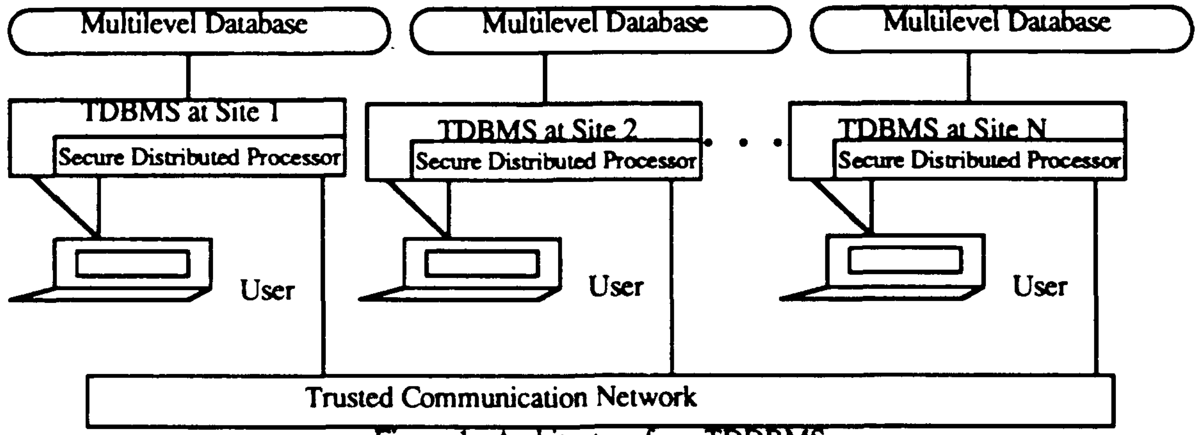


Figure 1. Architecture for a TDDBMS

The security policy for a TDDBMS will consist of a set of polices for mandatory security, discretionary security, integrity, and authentication among others. Our focus has been on a mandatory security policy for a TDDBMS. An effective mandatory security policy for a TDDBMS should ensure that users only acquire the information at or below their level. The basic mandatory security policy for the TDDBMS that we have considered has the following properties:

(i) Subjects are the active entities (such as processes) and objects are the passive entities (such as relations).
(ii) Subjects and objects are assigned security levels. The set of security levels form a partially ordered lattice (e.g., Unclassified < Confidential < Secret < Top Secret).
(iii) A subject has read access to an object if the subject's security level dominates the security level of the object.
(iv) A subject has write access to an object if the subject's security level is the security level of the object. (Note that this is a restricted version of the *-property of the Bell and LaPadula security policy [BELL75]).
(v) A subject S1 can send a message to another subject S2 if the security level of S2 dominates (i.e., is greater than or equal to) the level of S1.

In designing a secure system, it may be necessary for additional security policy extensions to be enforced. Such policy extensions are carried out by trusted subjects. That is, it has to be ensured that such a subject's security critical functions are carried out correctly. In addition, any subject that is privileged to violate the security policy must also be trusted. For example, if a message has to be sent from a Secret subject to an Unclassified subject, then the Secret subject must be trusted.

The security architecture that we consider is shown in figure 2. We assume that each node has a Trusted Computing Base (TCB). The TCB is the part of the host that enforces the basic mandatory security policy at that host. The Network TCB is responsible for enforcing the network security policy. The TCB hosts various trusted applications, such as a TDBMS and a SDP. Additional security policy extensions may be enforced by these applications, depending on their designs. In our design, the system must ensure that two DQPs (DUPs, DTMs, DMMs) at different nodes can communicate with each other only if they both operate at the same level.

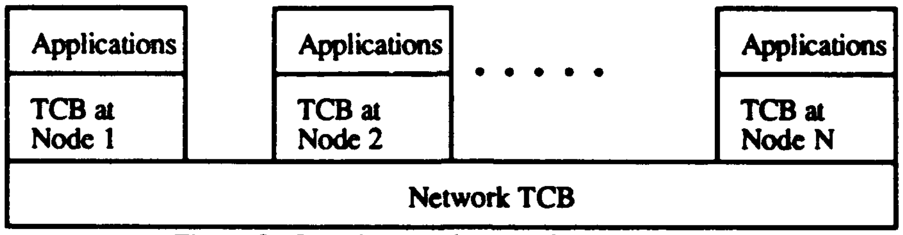


Figure 2. Security Architecture for a TDDBMS

2.2 QUERY PROCESSING

An important function of a TDDBMS is to provide the facility for users to query the distributed database system and obtain authorized and correct responses to the queries. Ideally, the distribution of the data should be transparent to the global users. That is, these users should query the distributed database as if it were a centralized system. The distributed query processor (DQP), which is responsible for handling queries, should determine the locations of the various relations involved and transmit the requests to the corresponding sites. The responses obtained from these sites have to be assembled before delivery to the user. In this section we describe the architecture of the DQP and then describe the design, simulation, and implementation of the join algorithms implemented. We have focussed mainly on the join algorithms because the join operation is time-consuming and has been studied extensively for nonmultilevel DDBMSs [CERI84].

2.2.1 Architecture of the DQP

The process architecture of the DQP is shown in figure 3. The components of the DQP are the Request User Interface Manager (Request-UIM), Response User Interface Manager (Response)), Query Transformer (QT), Query Optimizer (QO), and the Distributed Execution Monitor (DEM). The Request-UIM accepts a user's request and parses the request. The parsed request is given to the QT to decompose the request on a relation into requests on the various fragments. The decomposed request is given to the QO which generates an optimized execution strategy. The execution strategy is given to the DEM to carry out the execution. The DEMs at the various nodes communicate with each other via the trusted network in order to carry out the execution strategy. We assume that the network interface manager (NIM) which connects the DQP to the network is part of the trusted network. Finally, the DEM at the user's site delivers the response generated to the user via the Response. Next we discuss the impact of the security critical components of the DQP.

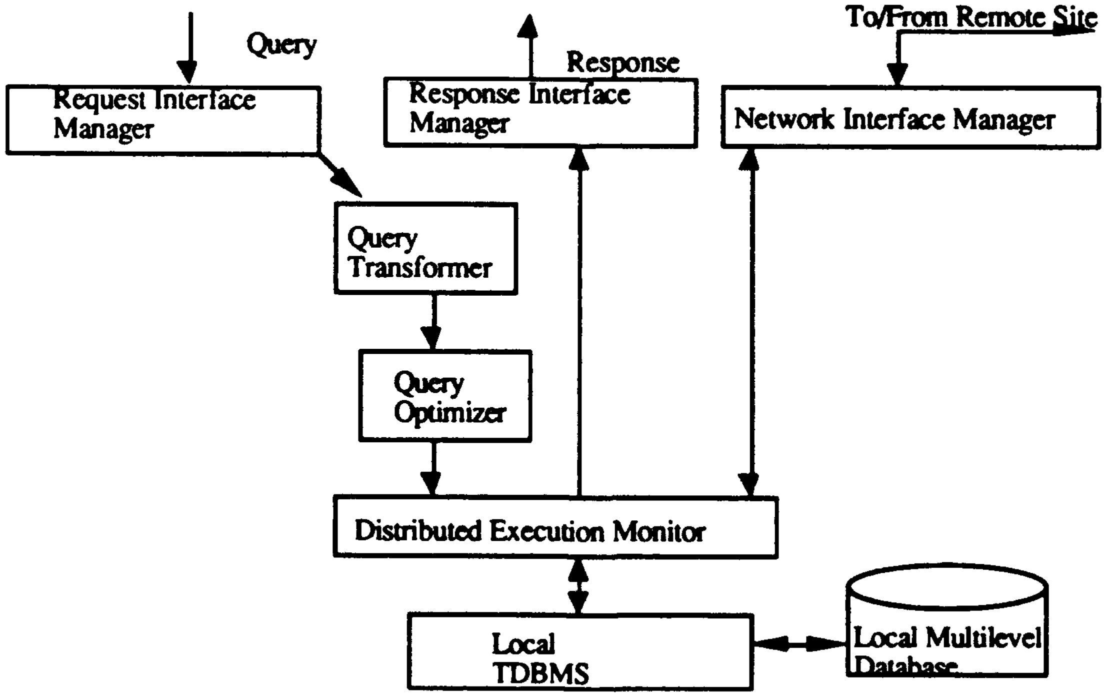

Query
Response
To/From Remote Site
Request Interface Manager
Response Interface Manager
Network Interface Manager
Query Transformer
Query Optimizer
Distributed Execution Monitor
Local TDBMS
Local Multilevel Database

Figure 3. Architecture of the DQP

Case 1: Untrusted DQP: Let us consider the case where the entire DQP is untrusted. In this case we assume that there is a DQP process per security level. When a user operating at level L issues a request, the DQP at level L is invoked by the trusted operating system to act on behalf of the user. The DQP at level L obtains the relevant metadata classified at or below level L in order to generate the execution strategy. The DEM component of the DQP, which carries out the execution strategy, acts as a user process to the TDBMS at that node. The TDBMS will ensure that the only data at or below the DEM's level is retrieved. The trusted network must ensure that the DEM operating at level L at a node N can only communicate with a DEM operating at at level L at a node M. Finally, the DEM at the user's site delivers the response assembled to the user. Since the DQP is untrusted, the labels displayed by the TDDBMS are advisory. In some cases this may not be a severe limitation if the labels displayed by the TDBMS at a node are themselves untrusted.[2] Another disadvantage with having an untrusted DQP is that one cannot guarantee the integrity of the data itself. For example, a user could request for the names of employees located at city 1 and the the query parser, transformer, or optimizer could modify the query to retrieve the employees located at city 2. An advantage of the having an untrusted DQP is that it has no security critical components.

Case 2: Trusting the entire DQP: At the other extreme, the entire DQP could be trusted. Then not only can it be guaranteed that the labels are trusted, the integrity of the results can also be ensured. However, this would mean trusting the SQL compiler well as all of the other modules of the DQP. As a result, all of the problems associated with having a large TCB will be present with this approach and it may be difficult to obtain high assurance with such an approach.

Case 3: Trusting the DEM and the Response User Interface: If trusting the labels is a requirement for an application, then the DQP must preserve the integrity of the labels (assuming that the labels displayed by the local TDBMSs are trusted). There are various ways to ensure this. In one of the approaches, all of the modules of the DQP except the DEM and the Reponse-UIM are untrusted. In this case, the DEMs can be guaranteed to preserve the integrity of the labels and the Response will guarantee that the labels accurately reflect the data. The trusted network must also ensure that the labels are not tampered with. Although this approach will significantly reduce the amount of trusted code than if the entire DQP were trusted, there is still a large part of the code that needs to be trusted. Furthermore, since the request-UIM, the query decomposer, and the query optimizer are untrusted, the integrity of the data returned cannot be guaranteed.

2.2.2 Algorithms

The DQP implements algorithms for the select-all operation as well as the join operation. Our focus was mainly on the join operation as it is the most time-consuming operation and has been studied extensively for nonmultilevel DDBMSs. We have simulated as well as implemented two join algorithms. The first algorithm is called a nondistributed join algorithm and the second one is called a distributed join algorithm. The difference between the two algorithms is the following. The nondistributed algorithm first merges the fragments of a relation classified at or below the user's level and then performs the join operation between the merged fragments using semi-join as the query processing tactic. During the merge operation, lower level polyinstantiated tuples are removed.[3] The distributed join algorithm first reduces the fragments classified at or below the user's level using the semi-join operation, joins the reduced fragments, and finally does the merge operation. The lower level polyinstantiated tuples are eliminated during the merge operation. Both algorithms are static. That is, the query execution strategy is generated first and then the execution is physically carried out. In determining an execution strategy, we used the cost model described in [CERI84].

A multilevel distributed database stored at two sites is illustrated in figure 4. We assume that all of the tuples of EMP-U and DEPT-U are classified at the Unclassified level and all of the tuples of EMP-S and DEPT-S are classified at the Secret level. In this database there is polyinstantiation across nodes. Figure 5 shows the join operation between the relations EMP and DEPT. In the restricted join operation the lower level polyinstantiated tuples are not included. In the unrestricted join operation, all of the tuples are included.

[2] Note that TDBMSs such as SeaView [DENN87] and Lock Data Views [STAC90] are designed in such a way that the labels are untrusted. Furthermore, TDBMS architectures such as OS-provided MAC Architecture and the Replicated Distributed Architecture do not ensure label integrity.

[3] Polyinstantiation [DENN87] is the mechanism that has been proposed to handle cover stories as well as prevent signalling channels in multilevel relational database management systems. We have assumed tuple level polyinstantiation where it is possible to have two different tuples with the same primary key at different security levels.

SITE 1

EMP1-U

SS#	Name	Salary	D#
1	John	20	10
2	Paul	30	20
3	James	40	20
4	Jill	50	20
5	Mary	60	10
6	Jane	70	20

DEPT1-S

D#	Dname	MGR
10	C. Sci	Jane
30	English	David
40	French	Peter

SITE 2

EMP2-S

SS#	Name	Salary	D#
3	James	70	20
7	David	80	30
8	Peter	90	40

DEPT2-U

D#	Dname	MGR
10	Math	Jane
20	Physics	Mary

Figure 4. Multilevel Distributed Database

SS#	Name	Salary	D#	Dname	MGR
1	John	20	10	Math	Jane
2	Paul	30	20	Physics	Mary
3	James	40	20	Physics	Mary
4	Jill	50	20	Physics	Mary
5	Mary	60	10	Math	Jane
6	Jane	70	20	Physics	Mary

Unclassified Join

SS#	Name	Salary	D#	Dname	MGR
1	John	20	10	C. Sci	Jane
2	Paul	30	20	Physics	Mary
3	James	70	20	Physics	Mary
4	Jill	50	20	Physics	Mary
5	Mary	60	10	C. Sci	Jane
6	Jane	70	20	Physics	Mary
7	David	80	30	English	David
8	Peter	90	40	French	Peter

Secret Restricted Join

SS#	Name	Salary	D#	Dname	MGR
1	John	20	10	C. Sci.	Jane
2	Paul	30	20	Physics	Mary
3	James	70	20	Physics	Mary
4	Jill	50	20	Physics	Mary
5	Mary	60	10	C. Sci.	Jane
6	Jane	70	20	Physics	Mary
7	David	80	30	English	David
8	Peter	90	40	French	Peter
1	John	20	10	Math	Jane
5	Mary	60	10	Math	Jane
3	James	40	20	Physics	Mary

Secret Unrestricted Join

Figure 5. Result of the Join Operation

2.2.2 Current Status of the Prototype

Figure 6 shows the configuration of the current prototype. The nodes are situated at MITRE's Bedford and Washington sites. Two of the machines are MicroVaxes running Ultrix (both products of Digital Equipment

Corporation). The local TDBMS in each of these machines is the Secure SQL DataServer (a product of Sybase, Inc.). The third machine is a SUN-3 running SunOS (both products of Sun Microsystems). The local TDBMS in this machine is the nonmultilevel version of Sybase DataServer (product of Sybase, Inc.). We implemented a front-end to this DBMS so that some multilevel security features of the Secure SQL DataServer were simulated.[4]

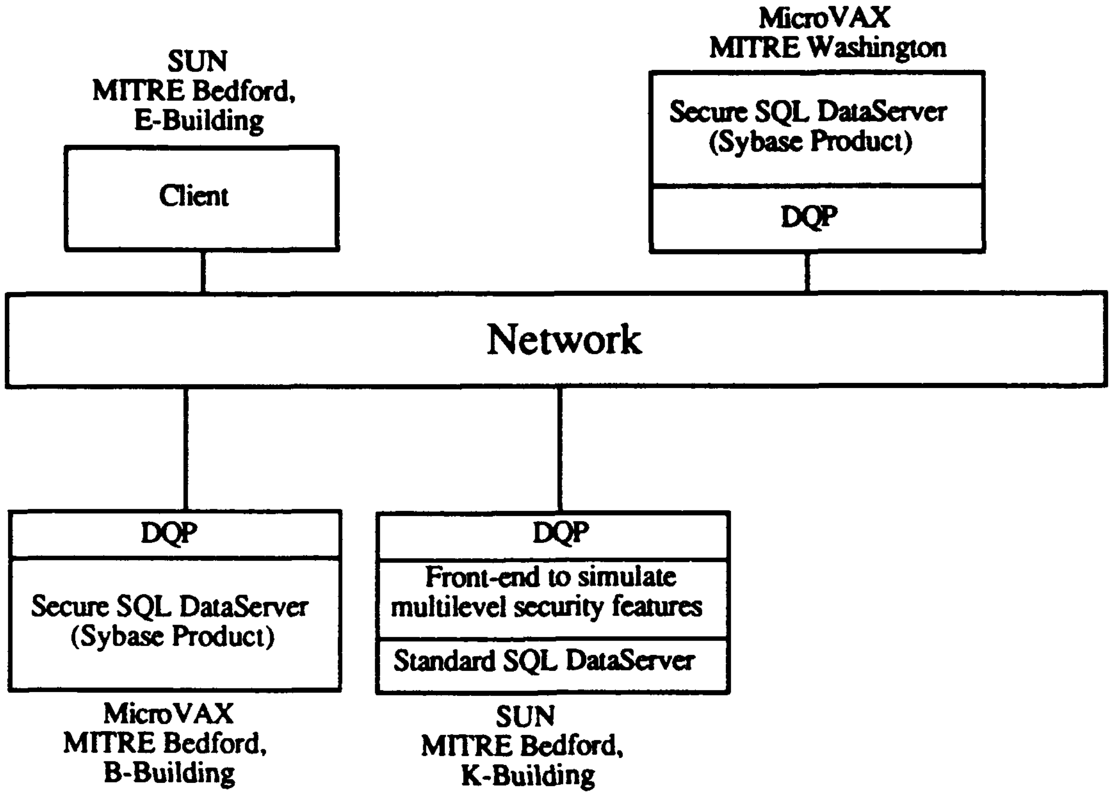


Figure 6. Configuration of the Current Prototype

During FY90 we implemented the DQP on the MicroVax and then ported it to the Sun-3. The details of this implementation are discussed in [RUBI90b]. During FY91, the Distributed Query Processor (DQP) was modified in order to establish communication between the Secure SQL DataServers at MITRE's Bedford and Washington sites. Specifically, the following modifications were made.

The first modification was made to accommodate the differences between the two Secure SQL DataServers when dealing with the name of the attribute which contains the security label. The Server at the Bedford site used the name "sl", while the Server at Washington's site refers to this as "sec_level". The second modification was made to handle the differences between the two Servers when returning the results of the SQL query of the form "select * from emp". One site would return the security label along with the attributes of EMP while the other would only return the attributes of EMP. We also had to modify the method that was used for a server to inform its existence to the other machines which either had a DQP or a client. In the earlier version of the DQP, the Unix[5] command, RCP, was used by the servers to accomplish this. RCP is a privileged command which may or may not be granted based on the user's host and the user's name. Since this privilege was not allowed between the servers at MITRE's Bedford and Washington sites, a program was developed to write the address of each server to a file replicated at each machine.

The prototype was demonstrated by placing a number of logical nodes at MITRE's Bedford and Washington sites to show that more than three severs could be used simultaneously. During a demonstration, the client could have been placed at either location.

---

[4] For a discussion on the Secure and Standard versions of the SQL Server we refer to [SYBA89]. While our design assumed that the network and the operating systems were trusted, those used in the implementation were not.

[5] Unix is a trademark of AT&T Bell Laboratories.

2.2.3 Simulation Results[6]

We simulated both the nondistributed and distributed join algorithms. We used the simulation language HGPSS developed by Rubinovitz [RUBI89]. A total of five experiments was carried out. Experiment I varied the number of tuples in the fragments and obtained execution times for the nondistributed algorithm. Experiment II varied the number of tuples in the fragments and obtained execution times for the distributed algorithm. Experiment III varied the number of sites and obtained execution times for both algorithms. Experiment IV also varied the number of sites and obtained the execution times for both algorithms. However, Experiment IV considered a very large number of sites while Experiment III considered a smaller number of sites. Experiment V varied the number of tuples and obtained execution times for both algorithms. Unlike Experiments I and II, Experiment V considered a very large number of tuples. The main purpose of Experiments I, II, and III was to compare the results obtained from the simulation to the results obtained from the implemented prototype. Overall we found that the simulation experiments matched the results obtained from the prototype fairly closely. A detailed discussion of the experiments are given in [RUBI92]. From these experiments we have arrived at the conclusion that in general, the nondistributed join algorithm gives better performance.

2.3 TRANSACTION MANAGEMENT/CONCURRENCY CONTROL

We have designed and simulated secure concurrency control algorithms. In this section we discuss the transaction model and the algorithms simulated.

2.3.1 Transaction Model[7]

In a multilevel environment, a distributed transaction as well as the associated subtransactions execute at the level of the user who requested its execution. At each node that the distributed transaction executes, there is an application agent. The agents operate at the same level as that of the transaction. The agents of the same transaction communicate with each other. One of the agents is called the root agent. When a "Begin Transaction" command is encountered, the root agent invokes the Distributed Transaction Manager (DTM) at the same node. This DTM, which acts as the coordinator, operates at the same level as the agent. The coordinator then issues the appropriate "Local Begin" command to its local transaction manager (LTM) and also communicates with other DTMs at the nodes in which the transaction executes in order to inform them of the "Begin Transaction" command. The DTMs at the other nodes communicate with their LTMs. Note that all of the DTMs as well as the LTMs operate at the same level as that of the transaction. When a "Commit" command is encountered, then the coordinator carries out the commit protocol. Figure 7 illustrates a model for distributed transaction management in a multilevel environment.

The security policy for the distributed transaction management extends the security policy for local transaction management. We assume the following policy for a centralized system.

- Each transaction is executed at the level of the user whose requests the execution. This level must dominate the level assigned to the program itself. (Note that a transaction is a program and could be assigned a security level.)
- A transaction does not change levels during its execution.
- A transaction reads from and writes into objects according to the mandatory security policy enforced by the system.

Extensions to distributed transaction management are the following:

- A distributed transaction executes at the level of the user who requested the corresponding application to be executed;
- A distributed transaction's subtransactions also execute at the same level;
- The subtransactions execute in accordance with the security policy enforced by the local system (it is assumed that all nodes enforce the same policy);

[6] Note that in general simulation experiments should precede prototype development. This is because simulation studies enable real-world scenarios to be depicted. As a result, one could select more suitable algorithms for implementation. However, in the case of the query processing algorithms, we implemented the prototype first before conducting the simulation experiments. It turned out to be useful because we were able to compare the simulation results with the implemented prototype for three of the five experiments.

[7] A transaction is a program unit which consists of a sequence of query and update requests. It must be executed in its entirety or not executed at all. Transaction management in a DDBMS involves the handling of distributed transactions. By a distributed transaction we mean a transaction which executes at multiple sites. The portion of the transaction which executes at a particular site is a subtransaction associated with that site.

- A distributed transaction (at the global level) reads and writes objects in accordance with the global mandatory security policy enforced (we assume that this is the same as for the local systems);
- A distributed transaction does not change levels during execution.
- Two DTMs at different nodes communicate only if they both operate at the same security level.

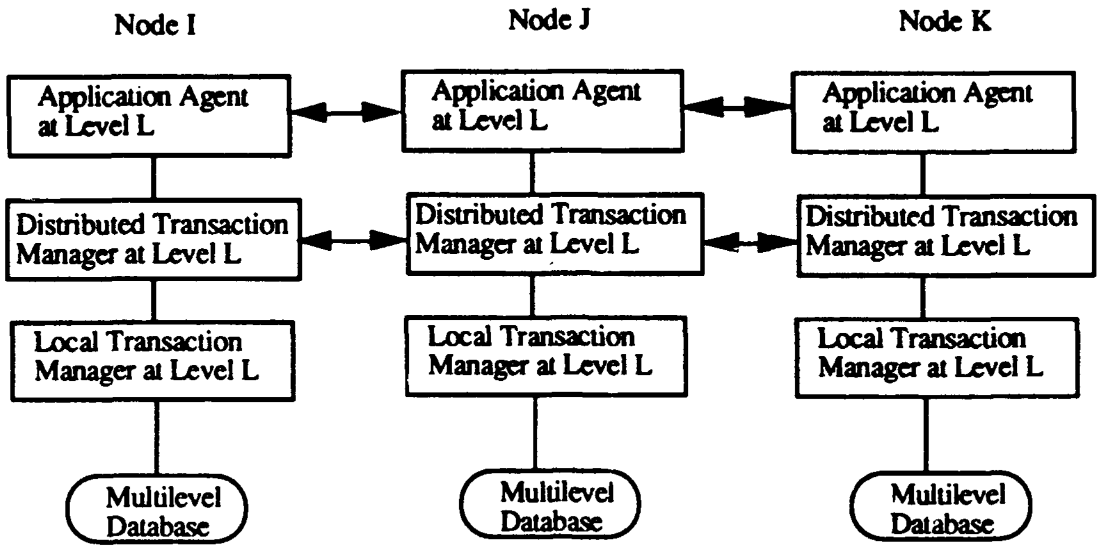

Node I
Node J
Node K
Application Agent at Level L
Application Agent at Level L
Application Agent at Level L
Distributed Transaction Manager at Level L
Distributed Transaction Manager at Level L
Distributed Transaction Manager at Level L
Local Transaction Manager at Level L
Local Transaction Manager at Level L
Local Transaction Manager at Level L
Multilevel Database
Multilevel Database
Multilevel Database

Figure 7. Multilevel Distributed Transaction Model

A major issue in transaction management is concurrency control. Concurrency control techniques ensure that the database is left in a consistent state when multiple, users attempt to update it at the same time. In a multilevel environment, in addition to consistency, it has to be ensured that high level transactions cannot affect the lower level ones at the local as well as the global levels. Concurrency control techniques must ensure that consistency as well as security has to be preserved. In our approach, there is a scheduler per security level. The scheduler at level L is responsible for scheduling transactions at level L. The concurrency control algorithms should ensure that the actions of higher level transactions cannot affect the lower level ones. At the same time the consistency of the distributed database must also be preserved. Concurrency control algorithms are discussed in the next section.

2.3.2 Concurrency Control

We have adapted some concurrency control algorithms to function in a secure environment. In this section we describe two of these algorithms that we simulated. One is the adapted locking technique and the other is the adapted timestamping technique. Below we provide a summary of these algorithms. A more detailed discussion is given in [RUBI92].

Locking

We have adapted the locking algorithm [ESWA76] for a multilevel environment as follows. To simplify the discussion we assume that there are two security levels, Secret and Unclassified. Since a Secret data object can be accessed only by Secret transactions, we are concerned with only Unclassified data objects. We assume that there are two copies (both Unclassified) of an Unclassified data object; one copy for Unclassified transactions and one for Secret transactions. For a data object X, let X1 be the copy for Unclassified transactions and X2 be the copy for Secret transactions. When Unclassified transactions request locks, the actions of the Secret transactions have no affect. The Unclassified transactions contend for locks on X1. As soon as an Unclassified transaction finishes updating X1, a new copy of X1 is created. Let this copy be X3. X3 will then be the new version of X for the Secret transactions. After an Unclassified transaction requests the write lock for X1, any read request on X by a Secret transaction would be directed to X3. X2 may be deleted later by a garbage collector. To ensure consistency, X2 should be deleted only if there are no locks queued for it. Since we assume that X2 is an Unclassified object, the process which deletes X2 must be trusted[8].

[8] If the garbage collector is untrusted, then it must be Unclassified. It will have no way of knowing whether the read requests of the Secret transactions are completed. Sufficient time could be given before deleting X2. However, there can be no guarantee that all the read requests on X2 are completed. A second solution would be to create the copy X2 for Secret transactions at the Secret level. Then a Secret process can delete

In distributed two-phase locking, each distributed transaction acquires all the locks before releasing any. We also assume that a transaction acquires all its locks before it starts execution. The locking technique is used together with two-phase commit for distributed transactions. In our work we have shown that the algorithm satisfies both the local and global serializability conditions [RUBI92].

Timestamping

We have adapted the timestamping algorithm [REED83] as follows Our objective is to ensure that higher level transactions do not interfere with lower level ones. In addition, the integrity of the database has to be maintained also. We propose the following modification to the timestamp algorithm. As in the case of the locking approach, we use two copies of an Unclassified data object; one for Unclassified transactions and one for Secret transactions. Whenever a Secret transaction issues a read request to an Unclassified data object X, first ensure that all Unclassified transactions with lower transaction numbers have completed their requests on X. Then process the read request issued by the Secret transaction. Note also that after an Unclassified transaction finishes updating an object, a new copy of the object is made for the Secret transactions. The old copy used by the Secret transaction may be deleted by a garbage collector. To ensure consistency, this copy must be deleted only after all Secret transactions with transaction numbers lower than that of the Unclassified transaction have finished the read operation on it. This means that the process which deletes it must be trusted.

The timestamping algorithm together with two phase commit is used for distributed transaction management. We have shown that both the local and global serializability conditions are satisfied.

2.2.3 Summary of Simulation Experiments

Twelve experiments were conducted. Experiments I to VI were for the locking concurrency control algorithm and experiments VII to XII were for the timestamping algorithm. For all experiments, the following two parameters were varied: transaction arrival rate and the number of objects accessed by each transaction. The transaction arrival rate varied for each set of experiments with an exponential rate with a mean of: 100, 200, and 300. The number of objects accessed was either 10 or 20. For all experiments the following parameters were fixed: the security level of each transaction was either Unclassified or Secret with an equal probability, the ratio of reads to writes were equal, the network configuration was identical, 50 transactions were allowed to complete. For each experiment, low, high, and average runtimes were gathered.

The results of the two sets of experiments seem to indicate that the timestamping algorithm, on the average, performs better than the locking algorithm. One reason for this might be caused by the time delay used to wait before trying to obtain the necessary locks before the transaction is allowed to continue. A slight delay of 200 was used which might have affected the overall run time of each transaction. Details of the experiments are given in [RUBI91d].

3. MIGRATING TO A HETEROGENEOUS PLATFORM

Various types of heterogeneity have been identified for a secure distributed environment [THUR91b]. We have focussed mainly on one type of heterogeneity which is handling different accreditation ranges. We discuss architectural issues, query processing, and transaction management for such an environment and then briefly identify some of the other types of heterogeneity that need to be addressed in the future.

3.1 HANDLING DIFFERENT ACCREDITATION RANGES

In this section we describe query processing in a limited heterogeneous environment. In particular, we consider the case where not all of the nodes handle the same accreditation ranges.

Figure 8 illustrates a nonreplicated architecture for an TDDBMS which handles different accreditation ranges. That is, the lower level data are not replicated at higher levels for security reasons. Such an architecture is more appropriate for an autonomous environment as each local node has control of its own data. In this example, the TDBMS at site 1 handles the range from Confidential to Top Secret while the TDBMS at site 2 handles the range from Unclassified to Secret. It is assumed that each node has a TDBMS and a SDP hosted on a

X2 after all read requests are completed. However, in this case, if there are N security levels which dominate the Unclassified level, then a copy of the Unclassified data object X needs to be maintained at each of the N levels. There must also be a way to create the copies at these higher levels.

Secure Operating System. The operating system and the local TDBMS ensure that subjects read objects at or below their level and subjects write into objects at their level. The modules of SDP include the DQP. The nodes are connected via a trusted network. The network ensures that there is two-way communication between processes at different nodes operating at the same security level. If the processes are operating at different security levels, then communication must be via a trusted process.

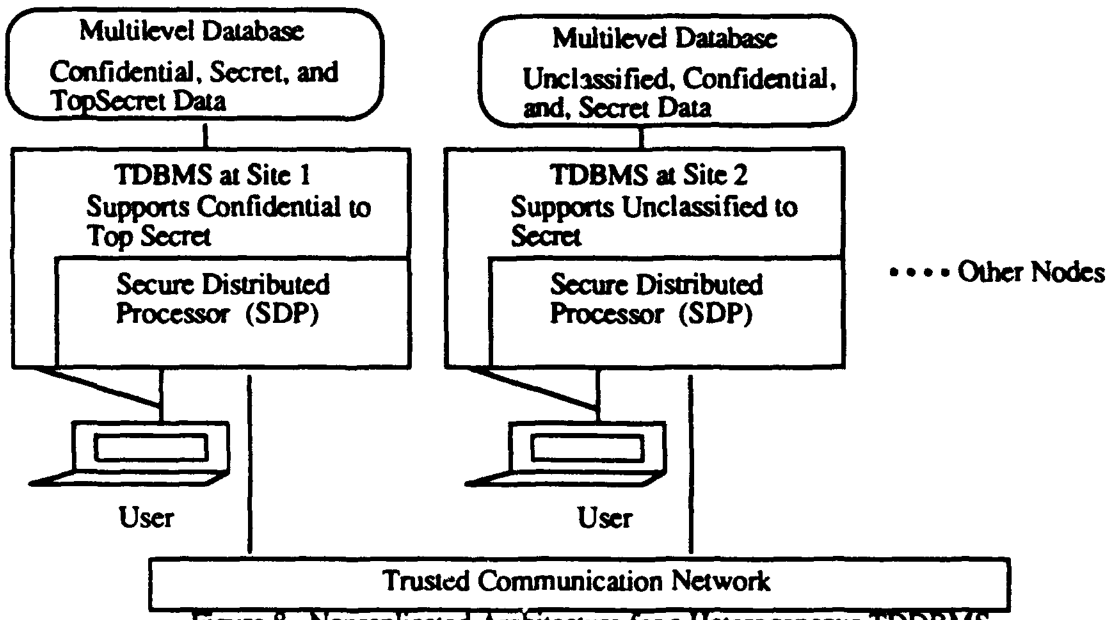

Multilevel Database
Confidential, Secret, and TopSecret Data
Multilevel Database
Unclassified, Confidential, and, Secret Data
TDBMS at Site 1
Supports Confidential to Top Secret
TDBMS at Site 2
Supports Unclassified to Secret
Secure Distributed Processor (SDP)
Secure Distributed Processor (SDP)
• • • • Other Nodes
User
User
Trusted Communication Network

Figure 8. Nonreplicated Architecture for a Heterogeneous TDDBMS

Since the lower level data is not replicated at a higher level, when a Top Secret user queries the system at site 1, the query has to be routed to site 2 which does not handle the Top Secret level. Therefore, the query must be routed via a trusted process which must ensure that it does not contain any Top Secret information. Since there is no data replication, update processing is less complex with this architecture as maintaining the consistency of the replicated copies is not an issue.

3.2. DQP FOR THE NONREPLICATED ARCHITECTURE

Figure 9 illustrates the software architecture of the DQP for the nonreplicated TDDBMS. In addition to the modules of the architecture of the DQP illustrated in figure 4, in figure 9, there is a module called the Query Trusted Process (QTP) which is between the DEM and the trusted network. QTP is a trusted process which must ensure that any sensitive portion of the query must be removed before it is transmitted to a remote DEM.

For example, consider the query, "Select all from EMP where Salary is greater than 50K" posed by a Top Secret user at site 1. Let us also assume that site 1 handles the range from Unclassified to Top Secret while site 2 is accredited to handle the range Unclassified to Secret. If the query has to be sent to site 2, the DEM at site 1 must remove the qualification clause from the query as it might contain potentially sensitive information. Note that DEM is an untrusted process and therefore it cannot be guaranteed to carry out its actions correctly. However, if it attempts to send any request to the DEM at site 2, the trusted network will reject it, as site 2 is not accredited to handle the Top Secret level. In other words, the NIM must be trusted to ensure that a Top Secret DEM does not communicate with a node which is not accredited to handle the Top Secret level. Therefore, the DEM at site 1 must send the query first to the QTP at site 1. QTP will check whether the qualification clause has been removed from the query, the string EMP is classified atmost at the Secret level, downgrade the request to the Secret level, and pass the request to the network via the NIM. The network then gives the request to the DEM at site 2 at the Secret level. The modified request will be to "select all from EMP". The Secret DEM at site 2 will process the request and send the result to the Top Secret DEM at site 1 via the QTP. The DEM at site 1 will then select those tuples from EMP where the salary is greater than 50K.

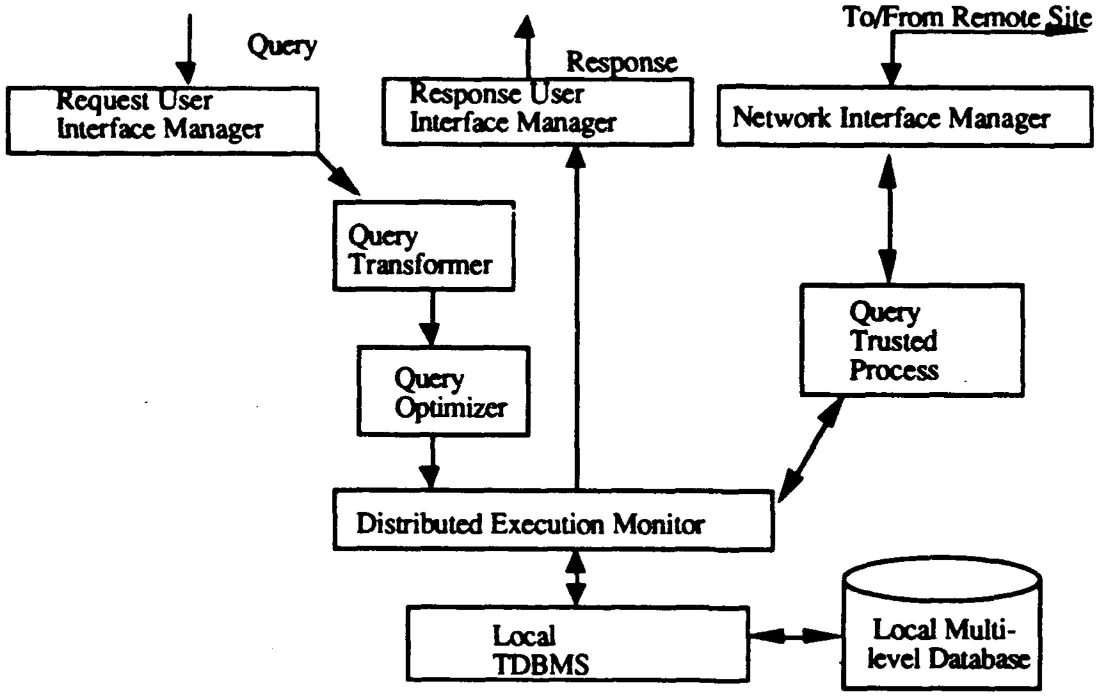

Query
Response
To/From Remote Site
Request User Interface Manager
Response User Interface Manager
Network Interface Manager
Query Transformer
Query Optimizer
Query Trusted Process
Distributed Execution Monitor
Local TDBMS
Local Multi-level Database

Figure 9. DQP for a Heterogeneous Environment

As discussed in section 2, if the entire DQP except the QTP is untrusted, then the integrity of the labels cannot be preserved. In order to preserve the integrity of the labels, the DEM and the Response-UIM must also be trusted. Note that if the DEM is to be trusted, then the functions of the QTP can be incorporated into those of the DEM. That is, a separate module for the QTP is not needed.

3.3 DTM FOR THE HETEROGENEOUS ARCHITECTURE

Issues on transaction management for a homogeneous environment were discussed in section 2.3. Figure 10 illustrates a transaction model for the limited heterogeneous environment that we have considered. It is assumed that there are four levels L1 < L2 < L3 < L4. Site 1 handles the range L2 - L4, site 2 handles the range L1 - L3, and site 3 handles the range L1 - L2. The DTMs at level L communicate with each other. However, since site 2 does not handle level L4, and site 3 does not handle the levels L3 and L4, the DTM at level L4 at site 1 communicates with the DTM at level L3 at site 2 and the DTM at level L2 at site 3 via a trusted process which we call the Transaction Trusted Process (TTP). Since site 3 also does not handle the level L3, the DTM at level L3 at site 1 communicates with the DTM at level L2 at site 3 via the TTP. We assume that there is such a TTP at each site.

As in the case of a homogeneous environment, we assume that a transaction is a program unit that is a series of query and update requests. A transaction may be distributed. That is, a transaction may be executed at several sites. We also assume that a transaction is assigned a security level and can be executed by a user whose level dominates the level of the transaction. However, if a transaction issued by a user at level L has to be executed at a site which does not handle the level L, then the subtransaction at that site is a read-only subtransaction. For example, if the execution of a transaction is issued by a user at level L4 at site 1, then that transaction can only read data items at sites 2 and 3. Therefore, sites 2 and 3 do not participate in the commit operation. That is, at sites 2 and 3 it is automatically assumed that such a transaction is committed. For the limited heterogeneous environment that we have considered, we have found that the locking algorithm, as we have described earlier, does not maintain consistency. The timestamping algorithm not only ensures security, but it also provides consistency. In this section, we discuses the issues for both algorithms.

Consider the locking technique for concurrency control. Let us assume that site 1 handles only the Unclassified level and site 2 handles both the Unclassified and Secret levels. Let X and Y be Unclassified data objects at sites 1 and 2 respectively. Since site 2 also handles the Secret level, there is a copy of Y at site 2 for the Secret transactions. Let Ti and Tj be two transactions executing at both sites 1 and 2. We assume that Ti is Unclassified and Tj is Secret. Since Tj is Secret, it can only read the data objects X and Y and it must originate at site 2. Furthermore, the subtransaction of Tj which executes at site 1 must be Unclassified and, therefore, any read request from Tj to site 1 must be via the TTP at site 2. Let Oi and Oj be conflicting operations by Ti and Tj respectively, on both X and Y. This means that Oi is a write operation and Oj is a read operation. The subtransactions of Ti and Tj that execute at site 1 are Ti1 and Tj1 respectively, and the subtransactions of Ti and Tj that execute at site 2 are Ti2 and Tj2 respectively. Note that Ti1, Ti2, and Tj1 execute at the Unclassified level and Tj2 executes at the Secret level.

At site 2, suppose Ti2 obtains a write-lock first on Y. If Tj2 then requests a read-lock on Y, it must wait until Ti2 releases the lock. However at site 1, suppose Tj1 obtains a read-lock on X first. If Ti1 requests a write-lock on X, since Tj1 is Unclassified, Ti1 must wait until Tj1 finishes. As soon as Tj1 finishes the read operation, Ti1 can obtain the write-lock on X. Ti1 and Ti2 can continue executing at sites 1 and 2, respectively. After Ti2 releases its write lock, Tj2 reads the new version of Y. Global serializability is not maintained because Tj precedes Ti at site 1 and Ti precedes Tj at site 2.

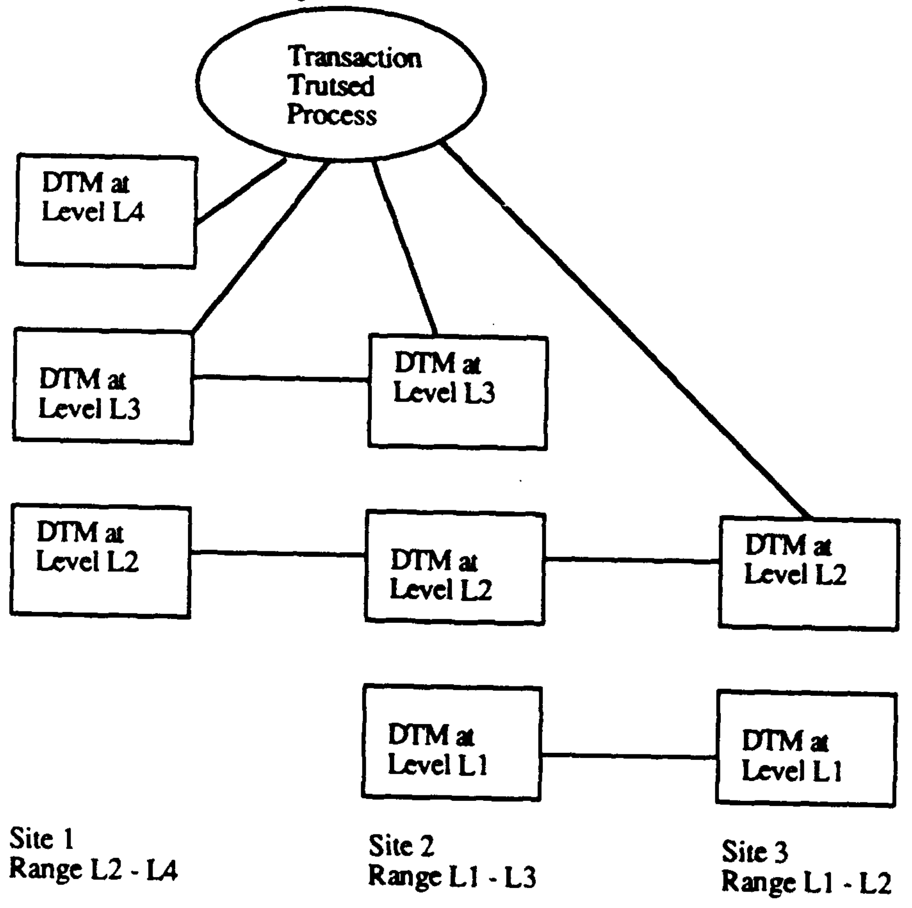


Figure 10. Model for Transactions

Next we discuss the timestamping algorithms for the same example. We consider two cases.
(i) Suppose the timestamp of Ti is less than the timestamp of Tj. When the subtransactions are assigned timestamps at sites 1 and 2, the TTP at site 2 can ensure that Tj1 is sent to site 1 after Ti1 is sent. This would mean that the timestamp of Ti1 is less than the timestamp of Tj1. Also the DTM operating at the Secret level at site 2 can ensure that the timestamp of Tj2 is greater than the timestamp of Ti2. Therefore, at both sites the operation Oi will precede Oj. That is, Ti will precede Tj.

(ii) Suppose the timestamp of Tj is less than the timestamp of Ti. If Ti1 is assigned a timestamp at site 1 before the TTP sends Tj1 to site 1, then Tj is aborted. Otherwise, Tj1 is assigned a timestamp less than the timestamp of Ti1. At site 2, the DTM operating at the Secret level can ensure that the timestamp of Tj2 is less than the timestamp of Ti2. If Oj does not precede Oi at site 1, then Tj1 is aborted. This would mean that Tj is not committed. Suppose Oj precedes Oi at site 1. If Oj does not precede Oi at site 2, then the only way this is possible is either: (a) for Tj2 to be aborted or (b) for Oj to be requested before Oi is requested, but Oi is carried out before Oj is carried out. However, Oj will be carried out on the older version of the data object and not on the version that results after Oi completes. The end result is as if Oj precedes Oi at site 2.

3.4. FUTURE CONSIDERATIONS

In order to ensure the interoperability of trusted DBMSs, in addition to handling different accreditation ranges, several other types of heterogeneity need to be addressed. In this section we describe some of the issues that need further investigation.

(i) Schema (or data model) Heterogeneity: Not all of the databases in a heterogeneous architecture are represented by the same data model. Therefore, the different conceptual schema have to be integrated. In order to do this, translators which transform the constructs of one multilevel data model into those of another need to be developed.

(ii) Transaction Processing Heterogeneity: Different TDBMSs may utilize different algorithms for transaction processing. The various concurrency control mechanisms that have been adapted to function in a multilevel environment need to be integrated for a heterogeneous environment.

(iii) Query Processing Heterogeneity: Different TDBMSs utilize different query processing and optimization strategies. One of the research areas here is to develop a global cost model for distributed query optimization.

(iv) Query Language Heterogeneity: Different TDBMSs will utilize different query languages. Standardization efforts are necessary to develop a uniform interface language.

(v) Constraint Heterogeneity: Different TDBMSs enforce different integrity/security constraints which may often be inconsistent. For example, one TDBMS could enforce a constraint that all employees are Secret while another TDBMS may enforce a constraint that all employees are TopSecret. These differences need to be reconciled.

(vi) Different Security Policies: Each TDBMS could enforce its own security policy for mandatory, as well as discretionary security. In addition, different authentication and integrity mechanisms may be used. For example, one system could enforce a read-at-or-below-your-level and write-at-your-level policy while another system could enforce read-at-or-below-your level and write-at-or-above your-level policy.

(vii) Different Granularity of Classification: Even if the relational data model is utilized at all nodes, the granularity of classification could be different. For example, one system could enforce classification at the tuple level while the other system could enforce classification at the element level.

From the above discussion it is clear that the steps to achieving secure interoperability are by no means straightforward, and we believe that some of them are impossible, given the current state-of-the-art both in DDBMS and TDBMS technologies. Therefore, until global solutions to interconnecting heterogeneous and autonomous nonmultilevel databases become available, security policies, models, and architectures have to be developed on a case-by-case basis. That is, customized solutions to meet the individual customer needs are presently needed.

4. SUMMARY

In this paper we have provided a summary of some the essential points of our work in trusted distributed database management systems (TDDBMS) R&D. We first described an architecture for a homogeneous TDDBMS, and discussed query processing and transaction management algorithms for such an architecture. Then we migrated to a heterogeneous platform and discussed architectural issues as well as query processing and transaction management in a limited heterogeneous environment. In particular, issues on handling different accreditation ranges were described. We also identified the types of heterogeneity that must be handled if different TDBMSs are to be interconnected successfully.

ACKNOWLEDGEMENT

We gratefully acknowledge the U.S. Air Force (Rome Laboratory) for supporting our work on trusted distributed database management systems under contract F19628-89-C-0001. We thank Joe Giordano for his support and encouragement.

REFERENCES

[AFSB83] Air Force Studies Board, 1983, Committee on Multilevel Data Management Security, *Multilevel Data Management Security*, National Academy Press.

[BELL75] Bell, D., and L. LaPadula, 1975, *Secure Computer Systems: Unified Exposition and Multics Interpretation*, Technical Report No: ESD-TR-75-306, Hanscom Air Force Base, Bedford, MA.

[CERI84] Ceri, S., and G. Pelagetti, 1984, *Distributed Databases, Principles and Systems*, McGraw Hill, NY.

[COLL90] Collins, M., H. Rubinovitz, and B. Thuraisingham, June 1990, *Secure Distributed Query Processing Strategies*, WP28891, The MITRE Corporation.

[DENN87] Denning, D. E., et al., April 1987, "A Multilevel Relational Data Model," Proceedings of the IEEE Symposium on Security and Privacy, Oakland, CA.

[ESWA76] Eswaran, K., et al., 1976, "On the Notion of Consistency and Predicate Locks in a Relational Database System," *Communications of the ACM*, Vol. 19, #11.

[GRAU85] Graubart, R. and K. Duffy, April 1985, "Design Overview for Retrofitting Integrity-Lock Architecture onto a Commercial DBMS," Proceedings of the 1985 Symposium on Security and Privacy, Oakland, CA.

[REED83] Reed, D., 1983, "Implementing Atomic Actions on Decentralized Data," *ACM Transactions on Computer Systems*, Vol. 1, #1, pp. 3-23.

[RUBI89] Rubinovitz, H., 1989, "A Simulation Language for Distributed Databases," Ph.D. Thesis, University of Connecticut, Storrs.

[RUBI90a] Rubinovitz, H., and B. Thuraisingham, August 1990, "*Secure Distributed Query Processor: Overview*," MTR 10969 Volume 1, The MITRE Corporation.

[RUBI90b] Rubinovitz, H., December 1990, "*Secure Distributed Query Processor: Implementation Details*," MTR 10969 Volume 2, The MITRE Corporation (a version to be published in the Journal of Systems and Software).

[RUBI92] Rubinovitz, H., and B. Thuraisingham, 1992, "Simulation of Query Processing and Concurrency Control Algorithms foe a Trusted Distributed Database Management System (a version submitted for publication).

[SHET90] Sheth, A., and J. Larson, September 1990, " Federated Database Systems for Managing Distributed, Heterogeneous, and Autonomous Databases," *ACM Computing Surveys*, Vol. 22, #3.

[STAC90] Stachour, P., and M. B. Thuraisingham, June 1990, "Design of LDV - A Multilevel Secure Database Management System," *IEEE Transaction on Knowledge and Data Engineering*, Vol. 2, #2.

[SYBA89] Sybase Inc., 1989, Secure SQL DataServer, Commercial SYBASE DataServer.

[THUR90] Thuraisingham, B., *Multilevel Security Issues in Distributed Database Management Systems*, MTP-297, The MITRE Corporation, July 1990 (a version published in Computers and Security).

[THUR91a] Thuraisingham, B., and H. Rubinovitz , December 1991, *Design and Implementation Enhancements to the Secure Distributed Query Processor Prototype to Function in a Limited Heterogeneous Environment*, December 1991, M91-86, The MITRE Corporation (a version to be published in Computers and Security).

[THUR91b] Thuraisingham, B., November 1991, *Security Issues for Federated Database Systems to Manage Distributed, Heterogeneous, and Autonomous Multilevel Databases*, M91-78, The MITRE Corporation.

SECTION 7

LIST OF ATTENDEES

Vicky Ashby
The MITRE Corporation
7525 Colshire Drive
McLean, VA 22102

703 883-6368
fax: 883-1397
ashby@mitre.org

Leonard J. Binns
National Security Agency, R232
9800 Savage Road
Ft. Meade, MD 20755-6000

301 859-4494
fax: 850-7166

Rae K. Burns
The MITRE Corporation
202 Burlington Road
Bedford, MA 01730-1420

617 271-3256
fax: 271-3816
rkb@security.mitre.org

Tom Garvey
SRI International, EK260
333 Ravenswood Avenue
Menlo Park, CA 94025

415 859-3486
fax: 859-3735
garvey@ai.sri.com

Joe Giordano
USAF Rome Laboratory
RL/COAC
Griffiss AFB, NY 13441-5700

315 330-2805
fax: 330-3911

Richard D. Graubart
The MITRE Corporation
202 Burlington Road
Bedford, MA 01730-1420

617 271-7976
fax: 271-3957
rdg@security.mitre.org

Ira Greenberg SRI International, EL235 333 Ravenswood Avenue Menlo Park, CA 94025	415 859-5390 fax: 859-2844 ira@csl.sri.com
Hilary H. Hosmer Data Security, Inc. 58 Wilson Road Bedford, MA 01730	617 275-8231 fax: same Hosmer@dockmaster.ncsc.mil
Cynthia Irvine Gemini Computers, Inc. P.O. Box 222417 Carmel, CA 93922	408 373-8500 Irvine@dockmaster.ncsc.mil
Sushil Jajodia ISSE Department George Mason University 4400 University Drive Fairfax, VA 22030-4444	703 764-6192 fax: 323-2630 jajodia@gmuvax2.gmu.edu
T. Y. Lin Department of Mathematics and Computer Science San Jose State University San Jose, CA 95192	408 363-0814 408 924-5121 TYLIN@calstate.edu
Teresa F. Lunt SRI International, EL245 333 Ravenswood Avenue Menlo Park, CA 94025	415 859-6106 fax: 859-2844 lunt@csl.sri.com
Bill Maimone ORACLE Corporation 400 ORACLE Parkway, 14th Floor Redwood Shores, CA 94065	415 506-6370 wmaimone@oracle.com

Louanna Notargiacomo The MITRE Corporation 7525 Colshire Drive McLean, VA 22102	703 883-5302 notar@mitre.org
Dick O'Brien Secure Computing Technology Corporation 1210 West Co. Road, E. Suite 100 Arden Hills, MN 55112	612 482-7443 obrien@sctc.com
Chip Paradise The MITRE Corporation 7525 Colshire Drive McLean, VA 22102	703 883-7815 fax: 883-1397 paradise@mitre.org
Ravi Sandhu ISSE Department George Mason University, 155 E Bldg 4400 University Drive Fairfax, VA 22030-4444	703 764-4663 fax: 323-2630 sandhu@sitevax.gmu.edu
Bhavani M. Thuraisingham The MITRE Corporation 202 Burlington Road Bedford, MA 01730-1420	617 271-8873 thura@security.mitre.org
Kioumars Yazdanian ONERA - CERT 2, Avenue Edouard Belin B.P. 4025 F-31055 TOULOUSE CEDEX France	+33-61-55-70-75 fax: +33-61-55-71-72 yaz@tls-cs.cert.fr